A Neural Network Based Strategy for Robot Navigation in Dynamic Environments

Thesis

A Neural Network Based Strategy for Robot Navigation in Dynamic Environments

Hua Meng

Thesis submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in Artificial Intelligence and Control Engineering

The Open University
Milton Keynes
U.K.

October 1993

Date of submission: 27th October 1993
Date of award : 22nd December 1993

Abstract

This thesis studies the problem of robot navigation in the presence of unexpected environmental changes, which include unknown static obstacles and moving objects with unknown trajectories. Throughout this work, neural networks, as a new technique, are used to develop the functional components, which constitute the proposed navigation strategy. The neural network based navigation strategy we propose follows a two-level hierarchy and operates by integrating three network components (planner, navigator and predictor). At the higher level, the *planner* generates a nominal path from the initial position to the goal among the fixed known obstacles. At the lower level, the *navigator* incorporates the *predictor* to refine the coarse path by taking into account unexpected environment changes to achieve on-line real-time guidance.

During this research, three neural network components were developed. The path planner was developed first by using a three-layer feedforward network to optimize the cost (collision penalty) function of a path. The first version of the navigator – Navigator-1 – was then implemented using a multilayer feedforward network in which steering commands for static obstacle avoidance were generated by directly converting sensor reading through the network. To enable the navigator to handle moving objects with unknown trajectories, on-line motion prediction was introduced. The predictor was developed using an Elman recurrent net. Following that, an enhanced version of the Navigator-1 – Navigator-2 – was developed using a structured network in which three sub-nets were used – two of the sub-nets were used to realise dynamic obstacle avoidance and static obstacle avoidance respectively, and the third sub-net was used to make final steering decision by reconciling the results from those two sub-nets. Finally, the overall navigation strategy was implemented in a simulation system. Simulations showed encouraging results. It demonstrates that the neural network based strategy is capable of achieving adaptive navigation in the presence of unexpected environmental changes.

Acknowledgements

Firstly, I would like to thank my supervisors Phil Picton and Jeff Johnson for all they have done to support me in my studies. Without their help, this thesis would not have been possible. I cannot thank them enough.

Many other people have also provided valued input to my research through discussions, commenting on my written work. I particularly wish to thank George Rzevski, Yongzai Lu, Neil Woodcock, Nigel Cross, Paul Margerison, Ranjit Makwana, Richard Murphy, Senino Holtier, and Stephen Potter for helpful hints, provocative questions and practical assistance.

Many thanks to Pat Dendy, Carole Marshall, Jennie Conlon and Margaret Barnes, the secretaries of Design Discipline, for all their help. Thanks also to all members of Design Discipline (past and present) for friendship and for contributing to a happy and creative working environment.

I thank my husband Yibing for his love and support. He helped me overcome many difficulties.

Finally, I wish to thank my parents for their love and encouragement during my study.

This work was supported by a studentship from the Open University.

Table of Contents

Chapter 1 Introduction 1

1.1 Motivation 1

1.2 Statement of the Problem 4

1.3 Proposed Solution 6

1.4 Organization of the Thesis 10

Chapter 2 Background and Related Work 13

2.1 Motion Planning 13

2.1.1 Avoidance of stationary objects with known positions 13

2.1.2 Avoidance of moving objects with known trajectories 18

2.1.3 Avoidance of stationary obstacles with unknown position 23

2.1.4 Avoidance of moving objects with unknown trajectories 25

2.2 Neural Network 29

2.2.1 Introduction to neural networks 30

2.2.2 Neural network models 31

2.3 Summary 41

Chapter 3 A Neural Network Path Planner 43

3.1 Introduction 43

3.2 Neural Computation for Path Planning 44

3.3 An Enhanced Path Planning Algorithm 48

3.3.1 Collision penalty function–Neural network representation 48

3.3.2 Energy and dynamical equations 50

3.3.3 The effect of temperature in the penalty function 53

3.4 Test Results 54

3.4.1 Test example 54

3.4.2 Discussion 58

Chapter 4 Neural Network Navigator-1 — Design and Implementation 60

4.1 Introduction 60

4.2 Neural Networks for Sensory Motor Integration 61

4.3 Neural Network Navigator-1 65

4.3.1 The neural network based local steering scheme 66

4.3.2 Navigator network architecture 69

4.3.3 Training 71

4.4 Test Examples 74

4.5 Summary 76

Chapter 5 A Neural Network Motion Predictor 78

5.1 Introduction 78

5.2 Neural Networks for Time Series Prediction 80

5.3 The Recurrent Network Motion Predictor 82

5.3.1 Architecture of the Elman net – Predictor network 83

5.3.2 A fast learning algorithm – Pseudo-impedance control 86

5.3.3 On-line prediction and learning 88

5.4 Test Examples 90

5.5 Summary and Discussion 96

Chapter 6 Navigator-2 — An Improved Version of Navigator-1 100

6.1 Revising Navigator-1 100

6.2 Structure of the Navigator-2 104

6.3 Building Sub-net for Static Obstacle Avoidance 105

6.3.1 Architecture of the static obstacle avoidance sub-net 106

6.3.2 Training 107

6.4 Building Sub-net for Dynamic Obstacle Avoidance 111

6.4.1 Constructing the forbidden regions 112

6.4.2 Architecture of the dynamic obstacle avoidance sub-net 113

6.4.3 Training 114

6.5 Building Sub-net for Decision-making 119

6.5.1 AND combiner 119

6.5.2 Architecture of the decision-making sub-net 120

6.5.3 Training 121

6.6 Summary 124

Chapter 7 Simulation Results 126

7.1 The Simulation System 126

7.1.1 An overview of the navigation strategy 127

7.1.2 Implementation of the simulation system 128

7.2 Simulation Results 131

7.3 Summary 137

Chapter 8 Conclusions and Future Work 139

8.1 Summary of the Research 139

8.2 Contributions 140

8.2.1 Using a neural network based strategy to solve the robot navigation problem in the presence of unexpected environmental changes 140

8.2.2 Improving a neural network based real-time path planning algorithm 141
8.2.3 Developing the neural network navigator for on-line local guidance 142
8.2.4 Developing a neural network on-line motion predictor 143
8.3 Future Work 144

References 148

Appendix A Constructing Forbidden Regions 156

List of Figures and Tables

Chapter one

Figure 1-1. Structure of a robot navigation system 8

Chapter two

Figure 2-1. Processing element 32
Figure 2-2. A Multilayer feedforward network 33
Figure 2-3. A Hopfield network 37
Figure 2-4. A Kohonen network 39

Chapter three

Figure 3-1. An example of the neural network collision penalty for a rectangular obstacle. 46
Figure 3-2. Network for collision penalty function 49
Figure 3-3. Path planning with two obstacles 57

TABLE 3-1 Connection weights and bias terms 56

Chapter four

Figure 4-1. Local grid 67
Figure 4-2. Architecture of the Navigator-1 70
Figure 4-3. Coarse coding 70
Figure 4-4. A example of a pair of training pattern. 71
Figure 4-5. Training curve and weight distribution 73
Figure 4-6. Trajectories of navigation 76

TABLE 4-1. Training patterns 72
TABLE 4-2(a). Hidden layer connection weights and bias terms 74
TABLE 4-2(b). Output layer connection weights and bais terms 74

Chapter five

Figure 5-1. A block diagram of an Elman net 84
Figure 5-2. Architecture of the Elman net predictor 85

Figure 5-3. Prediction of a circular motion 92
Figure 5-4. Prediction errors 92
Figure 5-5. Prediction of the movement of puching-chair 94
Figure 5-6. Prediction errors 94
Figure 5-7. The modified architecture of the predictor for chaotic case 95
Figure 5-8. Prediction of a chaotic time-series 98
Figure 5-9. Prediction errors of a chaotic time-series 99

Chapter six

Figure 6-1. The local grid 103
Figure 6-2. Structure of the Navigator-2 104
Figure 6-3. Architecture of the SOA sub-net 106
Figure 6-4. A example of a pair of training patterns 107
Figure 6-5. A example of creating a forbidden region 113
Figure 6-6. Architecture of the DOA sub-net 114
Figure 6-7. A example of a pair of training patterns 115
Figure 6-8. Architecture of the decision-making sub-net 120

TABLE 6-1. Training patterns of the SOA sub-net 108
TABLE 6-2. The necessary training patterns of the SOA sub-net 110
TABLE 6-3(a). Hidden layer connection weights of the SOA sub-net 111
TABLE 6-3(b). Output layer connection weights of the SOA sub-net 111
TABLE 6-4. Training patterns of the DOA sub-net 116
TABLE 6-5. The necessary training patterns of the DOA sub-net 117
TABLE 6-6(a). Hidden layer connection weights of the DOA sub-net 118
TABLE 6-6(b). Output layer connection weights of the DOA sub-net 118
TABLE 6-7. Training patterns of decision-making sub-net 122
TABLE 6-8. The necessary training patterns of the decision-making sub-net 123
TABLE 6-9(a). Hidden layer connection weights of the decision-making sub-net 123
TABLE 6-9(b). Output layer connection weights of the decision-making sub-net 124

Chapter seven

Figure 7-1. Control flow chart of the system 130
Figure 7-2. Trajectory-1: an ideal navigation case 133
Figure 7-3. Trajectory-2: a navigation without the planner 134
Figure 7-4. Trajectory-3: a navigation with unknown static obstacles 135
Figure 7-5. Trajectory-4: a navigation with an unknown moving obstacle 136
Figure 7-6. Trajectory-5: a complex navigation case 137

Chapter 1

Introduction

1.1 Motivation

Intelligent mobile robots have many possible applications in a wide variety of areas, from space exploration to hazardous material handling, from military tasks to aid for the handicapped. In almost all cases, these robots should be able to deal with unstructured environments and be able to make intelligent decisions autonomously. The performance of a robot, however, depends on its ability to execute motion. Therefore, motion planning forms one of the most basic and important branches of robotics research.

In its simplest form, motion planning is the problem of finding a path from a specified starting point to a goal point while avoiding collisions with a set of known obstacles. In unknown environments, however, the motion planning problem must be formulated as a navigation problem where the path is modified dynamically in response to the environmental information collected in real-time.

Much of the prior work on this topic is concerned with methods for path generating in fully-known environments. The assumption behind these methods is that complete information about the robot, stationary obstacles and moving obstacles is known *a priori* so that the path finding problem can be solved by preplanning a path through the obstacles and then executing it blindly. Realistically, however, the workspace may change unexpectedly, environmental uncertainties exist not only at planning time but also at

execution time. It should be clear that a robot that can deal with unpredictable changes in its workspace will be capable of performing complex tasks independently in real world. This thesis concerns the navigation of a mobile robot from the start position to the goal position in the presence of unpredictable changes to the environment.

As a typical example of (specifying) the navigation problem addressed in this thesis, consider a hospital or a factory floor, where a wheeled mobile robot is in charge of delivering food to different rooms, or delivering component parts to work areas. The robot must move towards its goal in an unconstrained environment inhabited by other moving objects with unknown trajectories. While traveling to its destination the robot must take care to avoid collisions with fixed unknown obstacles (such as chairs or boxes left in the middle of a room) and with moving objects (such as people or other robots) which enter or leave the working area. Moreover, the robot should be able to generate and execute a movement on-line and in real-time to avoid the object safely when the robot's sensor suddenly perceives a moving object crossing its path.

As can be seen from the example, the problem of navigating a robot in the presence of environmental uncertainty is much more complex than planning motion in fully-known environments. It is impossible to preplan a path through the obstacles and then execute it blindly, since unforeseen changes may happen to the working area at any time. In addition, the methods for generating paths are generally computationally intensive and consequently cannot be implemented on-line. Thus, the algorithms for path finding in fully-known environments are in general not sufficient for the problem of navigating a robot in uncertain environments.

Artificial Neural Networks – the study of massively parallel networks of simple neuron-like computing elements – have recently seen a resurgence of

interest in the robotics community (Horne, Jamshidi and Vadiee, 1990). They may provide a solution to the problem of navigating a robot in uncertain environments, since the power of neural networks hinges upon their adaptive capability in changing and noisy environments. The neural network technology offers three most important advantages over conventional techniques;

a) Neural networks can be trained to *learn* different relationships between variables regardless of their analytical dependence (Rumelhart and McClelland, 1986). Hence, it is appealing to attempt to solve various aspects of the robot control problem without accurate knowledge of the governing equations or parameters by using neural networks trained by a sufficiently large number of examples.

b) Learning can be enhanced by the associativity and generalization properties of the neural networks (Josin, 1988). Thus the neural network does not have to learn all the possibilities – it can *generalize* from a few cases.

c) Neural networks perform powerful computations stemming from their massive parallelism. This feature would allow a robot to be capable of *processing large amounts of information in parallel* and respond to a constantly changing environment in real-time.

Therefore neural networks, in principle, offer the greatest promise for adaptive robot navigation in the presence of unforeseen environmental changes.

This thesis presents a neural network based strategy to the problem of navigation with an emphasis on the treatment of unpredictable events that disrupt the execution of the motion plan. Throughout this work this new computational architecture – Artificial Neural Networks – will be utilized as a

powerful tool to develop the functional components of the autonomous navigation strategy. In this work, we are more concerned with the practical and efficient use of neural networks in engineering problems, rather than its direct relevance to mimicking the structural properties of the human brain.

1.2 Statement of the Problem

Our objective is to navigate a mobile robot from the start position to the goal position in the presence of unpredictable events that include unknown stationary objects and moving objects with unknown trajectories. In order to define the specific problem that is investigated in this thesis, we first present the conditions and constraints associated with the navigation problem and introduce some notations.

We only consider a two-dimensional Cartesian environment onto which a grid structure is superimposed. The position of the robot and any objects are defined within this grid, so that every square in the grid is either occupied or unoccupied.

The robot is under perfect control without any error. In other words, we will concentrate on the problem of the decision-making aspects rather than the issue of the low-level control. We assume that we always have a map of the workspace which provides information about all the fixed objects in the working area of the robot. At each sample interval, the local environmental information is supplemented by local sensors, which could be a vision sensor on board the robot. Thus, briefly stated, our task is to guide the robot from an initial position to a final goal position in a two-dimensional workspace while avoiding collisions with any of the obstacles by using sensor readings and the map of the workspace.

To define the navigation problem more precisely, we need to make the following assumptions concerning the robot and the environment:

- A1. The robot has a servo control system that is capable of following any reference trajectories without delays or errors.
- A2. Estimates of the robot state, including its position, orientation and velocity in the global coordinate frame, are available for navigation decisions at regular time intervals.
- A3. The sensors on the robot are capable of detecting each obstacle and sensing the locations of each moving object within the navigation environment.
- A4. The static obstacles are modelled by convex polygons, and a moving obstacle is assumed not larger than the robot and its size is defined by its radius and the location of its centre. Its upper velocity never exceeds the speed limit of the robot.
- A5. The minimum distance between any two obstacles in the environment is greater than a predefined margin.

Assumption A1 indicates that we are not concerned with low level control problems which deal with effects of the dynamics and kinematics of the robot and its motion. Assumption A2 follows from the assumed knowledge of the servo system capabilities. Assumption A3 means the robot has the sensing capability to determine the location of any object in its workspace. It releases us from dealing with the vision issue. Assumption A4 gives the representation of the navigation environment, and assumption A5 is a basic assumption on the environment which implies a feasible path exists to the goal.

Following the five assumptions we make a further assumption that the robot is considered to be a point moving at a constant velocity, since the general problem in which a robot is a polygon can be transformed into a simpler problem in which a robot is a point by expanding obstacles by the size of the robot while shrinking the robot to a point, as in Lozano-Perez and Wesley

(1979).

With the above assumptions, the problem of handling unpredictable events is then the major focus of this work. As stated in the discussion of a typical navigation above, the unpredictable events involve unknown stationary obstacles, which are not the fixed obstacles charted in the map, and moving objects with uncertain motions. While moving toward its goal, the robot must avoid collisions with stationary (known and unknown) obstacles, at the same time it must take special care to avoid the moving obstacles whose movements might lead to unforeseen collisions. The moving objects only become significant when they can potentially disrupt the robot, and at other times can be ignored. The unexpected moving objects have to be detected, classified as disruptive or not, and finally avoided, if necessary. Obviously, such a process should be running on-line and in real-time. Thus, the main problem is how to reconcile all of these demands in an autonomous navigation strategy design and how to realize the process in the strategy.

1.3 Proposed Solution

With regards to maximal "autonomy", we decided to have a central control unit which assigns only simple tasks specified by a qualitative requirements, and leaves the lower level decision-making to be performed autonomously.

Our approach follows a two-level functional hierarchy. At the higher level, a nominal path can be determined based on the *a priori* of knowledge of the workspace, since it is assumed that there is always a valid map available to provide the built-in geometric information about the workspace. At the lower level, a on-line guidance scheme refines the coarse path in real time by taking into account unpredictable events that disrupt the motion of the robot during navigation. In this way, the low-level guidance scheme would be much more

autonomous, and the high-level planner would be free from dealing with the environmental uncertainty. The proposed two-level functional hierarchy is an important consideration in face of the complexity of the planning tasks we want the robot to solve.

On the other hand, designing an autonomous guidance scheme is the major focus of this work. Guidance, sometimes referred to as local navigation, is the problem of manoeuvring around local objects while directed along a global planned path. In the presence of unpredictable events, however, it is difficult to design such an autonomous guidance scheme to achieve adaptive performance. Neural networks, as a new technique, are introduced to solve the problems under these circumstances. The guidance scheme we proposed is composed of two major processes: predicting the motion of moving obstacles and collision avoidance, running in parallel and in real time.

The process of prediction starts when a moving obstacle is detected and classified as disruptive (i.e. the distance between the moving obstacle and the robot is closer than a predefined threshold). The motion *predictor* keeps tracking the moving obstacle by predicting the movement/trajectory of the obstacle at next sample time. At the same time, in the collision avoidance process the *navigator* drives the robot to avoid collisions based on the sensor reading in the neighbourhood of the mobile robot and the motion prediction of the disruptive moving obstacle.

We envisage the navigation strategy developed in this thesis to be used in a robot navigation system with the conceptual structure shown in Figure 1-1. As already stated, in this thesis our concern is the planning and decision-making problem rather than the low-level servo control problem.

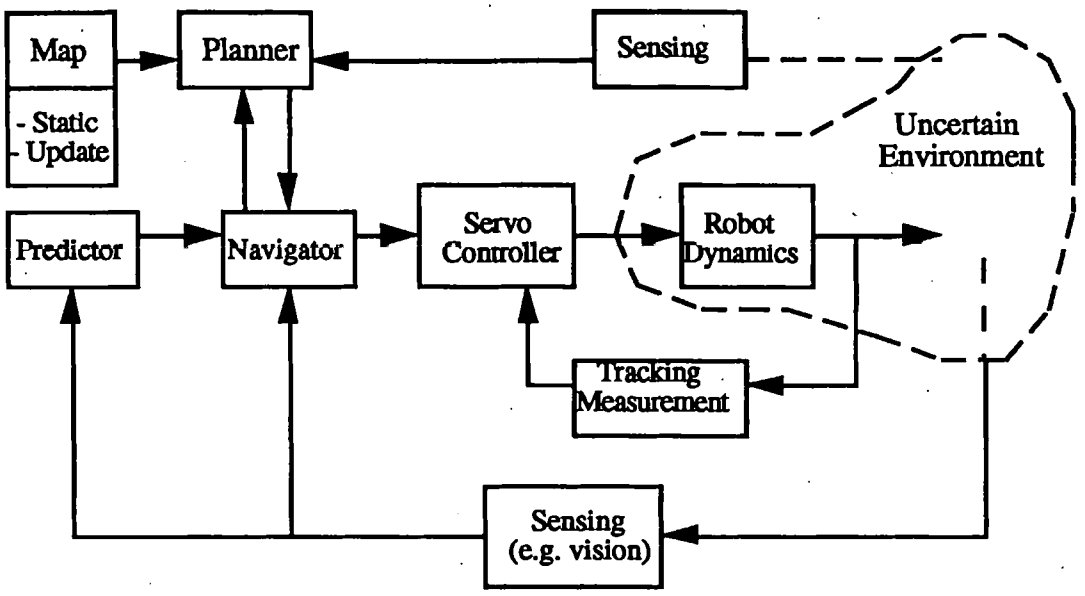


Figure 1-1. Structure of a robot navigation system

The overall method used in the proposed strategy is as follows. The process starts by loading the map of the workspace and taking sensor readings, which indicate where the objects are located initially.

The **Planner** is then called to plan a nominal path which is described as a sequence of via points (subgoals) from the initial position to the goal position among fixed known obstacles. Apart from the properties of general planners, the planner in this system should ideally have the capability of achieving real-time replanning in case the planner has to be called to handle situations that the navigator is unable to deal with because of the unpredictability of the environment. Most of the existing algorithms and methods for the path planning are generally computational intensive and cannot be implemented on-line. Neural computation, with massive parallelism, is therefore introduced for the real-time collision-free path planning.

When the planner returns a path, the two processes of predicting the motion of moving obstacles and collision avoidance start running in parallel.

In the prediction process, the **Predictor** keeps tracking moving objects by predicting the movement of the objects at next sample time. Predicting the motion of an obstacle is a very difficult task. This is because the attitude uncertainty characterizes the lack of knowledge about the motion of the obstacle and, the performance of the prediction requires on-line and in real-time. When a moving object is detected, the only information available to the predictor is the current location of the object, which is supplied by the sensor readings. The further data about the underlying movement of the object can only be acquired one-by-one through processing measurements obtained by sensing devices. Since the motion of the moving object varies with time, the predictor must be able to make a prediction within a short period while the moving object is classified as disruptive. This requires the predictor to make predictions on-line and in real-time by only using a very limited collection of the previous positions of the moving object. Furthermore, the velocity and rotation of the moving object are changeable at any time. The robot has no way of knowing how it will move. Therefore, the predictor should also be able to learn from experience and be adaptive to the behaviour of the moving object. The properties of recurrent neural networks make them obvious candidates for this difficult motion prediction task.

In the obstacle avoidance process, the **Navigator** generates steering commands to get the robot to each subgoal while avoiding collisions with unknown stationary obstacles and unforeseen moving obstacles using the sensor readings and the prediction of the moving objects. The navigator needs to make steering decisions in real-time by adaptively reacting to the environment to avoid unexpected collisions. There is a need to find an architecture that provides a very tight coupling between sensing and action to obtain minimal delay in responding to unpredictable changes in the environment. The architecture of multilayer feedforward type of neural networks provides a direct converter between sensing readings and action commands.

These three neural network based components are proposed to achieve the autonomous navigation strategy that can navigate a robot in the presence of unpredictable changes to the environment.

1.4 Organization of the Thesis

This thesis consists of eight chapters. In chapter two, we first survey the work relevant to this research, where attention is focused on the conventional approaches that are used in robot motion planning. The purpose of this review is to provide a context for the research reported here. Following that, neural networks, as a new technology, is briefly introduced. Finally, several typical models of neural networks are described in detail.

On the basis of the general knowledge about the neural networks, three neural net based components, which constitute the proposed navigation strategy, are presented in turn in the following chapters.

In chapter three, we focus on the design and implementation of the planner. In this chapter, we first briefly review previous research which applies neural network approaches to the problem of path planning. Based on the survey, we then develop a real-time path planning algorithm. In our method, a multilayer feedforward network is employed to derive the collision penalty function associated with the obstacles in the workspace. A collision-free path can be obtained by minimizing the energy of a path in terms of the path length and the collision penalty. The behaviour of the planner is finally examined in the simulation.

In chapter four, the first version of the navigator is presented, in which the navigation environment is simplified by assuming that only stationary

obstacles exist. In this chapter, again, we first briefly review the neural network based approaches for sensory motor integration, and then develop a local steering scheme based on a feedforward network for the navigator, in which steering commands are generated by directly converting sensor readings through the network. Finally, we present test examples to examine the behaviour of the navigator.

In chapter five, we focus on the motion predictor. We first review recent progress on using neural network approaches in the area of time series prediction. We then present the predictor which is based on a recurrent network and a fast version of the backpropagation learning algorithm. Following that, we present an on-line prediction and learning scheme, which is developed for the predictor to adopt uncertain changes in the movement of a object. Finally, the predictive capability of the predictor is examined by analysing of the simulation results.

In chapter six, the improved version of the navigator is developed so that the navigator can deal with much more complex situations where moving obstacles are involved. We first give some revisions which are necessary to adopt the previous navigator in the presence of time-varying conditions. We then introduce the implementation of the revised navigator that consists of a structured network, in which two sub-networks are used to realize dynamic obstacle avoidance and static obstacle avoidance respectively, and the third sub-net is used to make collision-free steering decision by reconciling the results from those two sub-nets.

In chapter seven, a simulation system which combines three components (planner, navigator and predictor) is firstly described. The purpose of building this system is to test the effectiveness of the navigation strategy we proposed, and demonstrate the integration of the three components. We then show

several simulations, in which different unexpected events are present. Finally, we summarize the simulation results.

We conclude in chapter 8 by summarizing the contributions of this thesis and exploring the future direction of this research.

Chapter 2

Background and Related Work

In this chapter, we review the relevant work to our research and briefly introduce neural networks which are the basis for our work. The purpose of this review is to provide a context for the research reported here. In this chapter, we mainly focus on conventional approaches that are used in motion planning. We defer the discussions of approaches using neural networks until chapter 3, 4 and 5 where the techniques that we used will be described in more detail.

2.1 Motion Planning

Loosely stated, the fundamental task of motion planning is to move a robot from a starting configuration to a goal configuration without colliding with any objects in the workspace. Our work addresses a variation of the motion planning problem, which entails a dynamic environment that involves unknown stationary obstacles and unexpected moving obstacles. It therefore seems natural to examine the relevant work from two perspectives:

1) whether the obstacles have fixed positions or move,

2) whether the motions of moving obstacles are known or not.

The following description is organised around this view.

2.1.1 Avoidance of stationary objects with known positions

An environment with stationary known obstacles is the simplest type of

environment in which to plan motion, and algorithms for this problem are often used as ingredients of Motion Planning (MP) algorithms for other environments. Because the environment is fully-known, the problem is best solved by a pre-planning approach: a full plan is developed and then executed blindly by the robot.

Work in this area can be divided into theoretical work and practical studies. In theoretical studies the objective is to develop exact algorithms, and the problem of moving a rigid robot through an environment with stationary known objects is at times referred to as the Piano Movers' Problem (Sharir, 1989). Practical studies are concerned with developing heuristic algorithms which are aimed at finding a solution, but not necessarily the best one. A survey of the different approaches can be found in Sharir (1989).

The main algorithms for moving a robot in the presence of stationary known obstacles can be classified as: skeleton approach; cell decomposition approach and potential-field approach. In the following paragraphs, we describe them in turn.

a) Skeleton Approach

In this approach, the free configuration space is retracted to a network of one-dimensional lines. The search for a solution is limited to the network, and the MP problem becomes a graph-searching problem. The well-known skeletons are the *visibility graph* and the *Voronoi diagram*, which are commonly used in 2-D situations.

Note that the concept of the configuration space (Cspace) is introduced by Lozano-Perez and Wesley (1979). A configuration of an object or a robot is a set

of independent parameters completely specifying the position of every point of the object or robot.

To plan motions for a translating polygonal robot among polygonal obstacles, configuration obstacles are computed in Lozano-Perez (1983). The basic idea is that of "shrinking" the finite-size robot to a point and "growing" the obstacles accordingly. Through this "shrinking" and "growing" transformation, the geometrical properties of the obstacles and robot are preserved. The problem of finding a path for a finite-size robot, therefore, can be reduced to the problem of finding a path for a point. Within this free Cspace a *visibility graph* is then computed using vertices of the polygon obstacles plus the given start and goal points. The shortest path is found by searching the visibility graph. However, when working with complex shaped obstacles the search space becomes so huge that the computation becomes non-trivial.

The *Voronoi diagram* and its variations are also used for planning a path for a disc robot among stationary obstacles (O'Dunlaing and Yap, 1985). The Voronoi diagram is a planar network of line segment and parabolic curves, which are the set of points equidistant from at least two obstacles. In contrast with the path in the visibility graph along which the robot gets close to the obstacles, the path generated by the Voronoi diagram is much safer. This is because it keeps the robot as far as possible from the obstacles.

b) Cell Decomposition Approach

In this approach, the free Cspace is decomposed into a set of simple cells, and the adjacency relationships among the cells are computed. A collision-free path between the start and the goal configuration of the robot is found by first identifying the two cells containing the start and the goal, and then connecting

them with a sequence of connected cells. Examples of object-independent cell decompositions are the *grid* and the *quadtree*.

Gouzenes (1984) proposes the use of a *grid* to divide free space into a set of adjacent cells. The cells are developed using a variant of Voronoi diagrams, where the distance to an object determines the cell in which a point lies. A node is picked in the centre of a cell, and a path is developed by connecting those nodes. The plan is executed by moving from node to node. In addition Gouzenes proposes using Khatib's potential field (see next section) to navigate locally from one node to the next. This helps when executing fine motion.

A quadtree-based algorithm for a point robot is present in Kambhampati and Davis (1986). The nodes of the *quadtree* are either black (obstacle occupied), white (free space), or grey. A cell which contains only a few small obstacles is denoted as a grey cell. A path is planned using the corners of white nodes as via points. Traversal from one corner to the next is made on the edge of the node quadrant, or across the diagonal. When a grey cell is selected as a part of an optimal path, it is refined into black and white cells. Search on a quadtree is much faster than that on a grid due to a small number of cells. With nonpolyhedral objects, however, a rather deep quadtree may be required.

Based on the cell decomposition representation, Jahanbin and Fallside (1988) develop a search algorithm by using the wavefront propagation method to find the safest and shortest path in a known environment. The approach is based on the principle in optics which states that every point on the current wavefront can be considered as a new secondary source. The Cspace is first decomposed into a set of simple cells (quadtree or array), each cell is given a *score* in terms of its occupation state. A path is found by starting at the goal cell and moving a 3 * 3 local operator (window) to its neighbour with the lowest score until the start cell is reached. We use a similar approach by organising

the local operator (information) as a local grid. A binary value is assigned to each cell in the local grid to indicate the occupation status.

The practical cell decomposition algorithms encompass a wide varieties of cell types. Brooks (1983) represents a 2D free space as a union of possible overlapping *generalised cones*. The axes of these generalised cones determine the natural free-ways for moving the object through the space. The algorithm translates a polygonal moving body along the axes or spines of the generalised cones and rotates it at the intersections of the generalised cones. This algorithm is fast and yields paths that avoid obstacles generously. However, there are two potential problems with this algorithm: the path found may not be short if the free space is wide, and the algorithm may not find a solution when the robot has to translate and rotate simultaneously to avoid obstacles.

c) Potential-Field Approach

The idea of using potential functions for obstacle avoidance was used by Khatib (1985). Khatib proposes that a collision-free path can be developed much faster, if done by the robot control system, using a retraction type of boundary following strategy rather than a planning system. This approach constructs a scalar function called the potential that has a minimum, when the robot is at the goal configuration, and a high value at obstacles. Everywhere else, the function is sloping down toward the goal configuration, so that the robot can reach the goal configuration from any other configuration by following the negative gradient of the potential. The high value of the potential prevents the robot from running into obstacles. Although such an algorithm may not be exact, it is attractive due to its low computational costs. Therefore it is often used as a submodule for global motion planners to solve local problems.

The worst problem with the potential-field approach is that local minima in the field can be present at places other than the goal point. These spurious local minima often trap the robot. Another disadvantage of this approach is that the expression for the potential becomes very cumbersome when there are many concave objects.

The simplicity of the potential-field concept has led to extensive research in this area. Originally it was proposed to repel a robot from a set of objects. Arkin (1987) views this as being only one of many behaviours which can be implemented with potential fields. He defines several potential functions which implement behaviours such as wall following. He proposes combining several of these potential functions into the total field to implement more complex behaviours.

On the whole, the major deficiency of the approaches described so far with respect to the motion planning is that they assume the environment is fully known and static. In the next subsection, we look at the problem of avoiding moving objects with known trajectories.

2.1.2 Avoidance of moving objects with known trajectories

In this situation, a robot is required to move among a set of obstacles which are moving through space. The trajectories of the objects are fully specified. Similar to the problem of avoiding stationary objects with known positions, as we described in the last subsection, this problem can be solved by preplanning a trajectory through the objects and then executing it blindly.

Those approaches at dealing with stationary objects, however, are not always applicable to the problem of moving obstacles. For example, since visibility is constantly changing as obstacles move, the *visibility graph* method cannot

directly apply to the situation involving moving obstacles. Most cell decomposition methods divide space into some type of smaller space, and the assumption behind these methods is also that the geometry is invariant during the path search process. Therefore, applying the methods in the subsection 2.1.1 to the problem involving moving obstacles is generally not straightforward.

Planning a motion among moving obstacles is intrinsically harder than planning a motion among stationary obstacles. Reif and Sharir (1985) show that the problem of moving a three-dimensional rigid robot among moving obstacles is PSPACE-hard when the velocity of the robot is bounded, and NP-hard without such a bound. The problem of moving a convex polyhedron robot with a bounded speed among convex polyhedron obstacles moving with constant velocity and without rotation is referred as the *asteroid avoidance* problem.

Reif and Sharir (1985) suggest an algorithm for this problem. In the two dimensional case, they consider a three dimensional configuration space-time. In this Cspace a sequence of three kinds of actions can be constructed, yielding a boundary-following path. These actions consist of either straight line movements, intervals where the robot stays stationary in space, and movements where the robot moves along the boundary of an object. The time bound is derived by counting the number of actions necessary.

Canny and Reif (1987) use a cell decomposition method to construct a path. They construct a system of polynomials which is satisfiable if and only if there is a successful sequence of straight line motions defining a path. They then use what they call the Roadmap Algorithm to find a skeleton of the set of points. The skeleton then supplies the desired path. They show that motion planning for a point robot in 2D environment with a bounded velocity is NP-hard, even

when the moving obstacles are convex polygons moving at constant linear velocity without rotation.

Kant and Zucker's (1986) path-velocity decomposition approach is probably the first practical algorithm for motion planning among moving obstacles. They decompose the problem into two sub-problems: the Path Planning Problem among stationary obstacles (PPP) and the Velocity Planning Problem along a fixed path (VPP). For the first sub-problem, they plan a path among stationary obstacles while ignoring all the moving obstacles. For the second sub-problem, the speed of the robot along the path is adjusted by looking at the trajectories of the objects as they cross the path in time. For each time instant, the points in the path occupied by obstacles are computed and represented as polygonal obstacles in the space-time configuration space. A graph called the *directed-graph* is constructed by using the vertices of these polygons to define regions. These regions can be thought of as objects to be avoided. The path in this new space defines the velocity profile of the robot. When the results of the original path around the static objects and the new path around the transformed moving objects are combined, they define a path which avoids all obstacles. Kant and Zucker, however, point out that the method is not capable of handling some types of object trajectories, such as an object tracking the robot. The system suffers because it cannot modify the original trajectory once it is chosen.

Moving a point robot among translating polygonal obstacles is considered in Fujimura and Samet (1989). The polyhedral obstacles in the 3-dimensional space-time is represented with an octree. Then Winston's A* search, described by Latombe (1991), is used to find a shortest path. Since the octree representation can change significantly even for small changes in the space-time obstacles, the uncertainty in the velocities of obstacles may result in different paths. In their later work, Fujimura and Samet (1990) develop the

concept of accessibility to determine a time-minimal path for a moving point in a 2D time-varying environment filled with convex polygonal obstacles that move at constant speed. Under the assumption that the moving point is able to move faster than any obstacles, they show that the time-minimal path consists of a sequence of edges in the 'accessibility graph'.

Shih, Lee and Gruver (1990) propose a method that finds a collision-free, near-optimal path among polyhedra in the space-time domain. The method is based on a graph search technique and linear-programming. The methodology can be extended to motion planning in higher dimensional spaces. However, the search space will increase significantly when the number of obstacles becomes large.

Pan and Luo (1990) develop the time-varying half-planes representation to model moving polygons in 2D space. In their method, a point robot is considered by growing obstacle boundaries. Static obstacles do not change their positions during robot operations while moving obstacles can translate along a predefined trajectory. They use the concept of traversability vectors to analyse the spatial relationship between the robot and moving obstacles. Given a predefined path, the occupancy of the path by moving obstacles can be detected and registered on the constraint map. A search algorithm is then developed to coordinate the robot motion.

In their later work, Pan and Luo (1991) extend their previous work by developing an efficient interference detection scheme that deals with polygonal obstacles moving with translation and rotation at non-linear velocity. The scheme works directly in a two-dimensional space without reducing the robot to a point. Moreover, obstacles are not restricted to be convex. Thus, the improved method is more accommodating to the dynamic motion planning problem. Compared with Kant and Zucker's path/velocity

decomposition approach, they also suggest that it would be necessary to test several path alternatives to determine the minimum-time path. However, the limited predefined path alternations still lead their approach not give a solution even though one may exist.

Again, all the methods we have described so far assume *a priori* knowledge of the motions or trajectories of the moving objects. It is difficult to extend these results to the situation where the trajectories of the objects are unknown and still guarantee that a solution will be found whenever a solution exists.

A special case of the avoidance of moving objects with known trajectories is the problem of coordinating the motions of multiple robots. This category of problems is usually called *coordinated motion planning*. Since we can control the trajectories of all the robots, the problem is not so difficult as the case where we must consider moving obstacles with unknown trajectories. Thus, we only give a brief explanation to this special problem below.

The main difficulty of this form of planning is that other robots are moving in the same workspace. We therefore must plan paths for all the robots to coordinate their movements. In most of the work, the coordination can be done in two ways: robot communication or priority assignment.

In robot communication, the robots plan their motion independently. If a collision between two robots is detected, the two robots then negotiate with each other for the right of passage. The winning robot proceeds with its plan, and the losing robot must replan. Tournassoud (1986) uses this scheme to plan motions for multiple robots. In his method, the workspace is divided into adjacent, disjoint regions. This is done by using tangent lines from one robot to another robot. Each robot can move along the boundaries of the regions without the risk of colliding with other robots. Eventually the robot may have

to exit the safe region. It then communicates with the other robots to decide where best to locate the tangent lines for the next set of movement. This is done by minimising the distance from each robot to its goal.

Priority assignment is a much simpler scheme to implement. In this scheme, there is a supervisor which controls all the robots. Again each robot can plan its own path. If a collision is predicted, the supervisor is then notified. It decides which robot has the priority, and that robot is allowed to proceed. The other robots replan their paths. Erdmann and Lozano-Perez (1987) describes a system based on a priority assignment scheme. In their system, the space is "sliced" into time slices. In each of these time slices the higher priority robots are considered stationary, and a path is planned using a method for static environment. A path is then planned from one time slice to the next by connecting robot vertices between slices.

2.1.3 Avoidance of stationary obstacles with unknown position

In this case, the robot is moving through a workspace which contains fixed unknown objects. It is therefore impossible to preplan, since the positions of some of obstacles must be discovered during execution. When a robot explores its workspace, the sensors fitted on the robot supply it with the information about the obstacles in the neighbourhood of the robot. The robot's path is constructed on-line, point by point, using the local description of the scene. In such unknown environments, the sensor-based motion planning problem is formulated as a navigation problem.

If the position of the obstacles are totally unknown initially, the robot must explore its workspace and develop a map of the obstacles in order to plan a path to its goal. With polyhedral objects the map is usually represented by a visibility graph which is a graph of the vertexes of the known polyhedral

objects. Visibility graphs are used in Oommen et al's work (1986). They use boundary following to move a robot through unknown terrain. The robot uses its sensors to find the obstacles in the workspace, and develops a visibility graph as it moves.

When the positions of objects are mostly known, an avoidance system might do gross preplanning, and modify its actions by taking into account new information while executing the plan. This is the approach used by Cheung and Lumelsky (1989) for a manipulator. In their system, The manipulator is coated with an array of proximity sensors. As the manipulator executes a path, the proximity sensors inform the system if the arm is approaching any objects, and if so local evasive action is undertaken. This involves moving away from the object, following the tangent to the objects surface.

A major contribution to solving the navigation problem using the algorithmic approach is Lumelsky and Stepanov's work (1987). They approach the navigation problem with the assumption that only local information about the environment is available. They consider the navigation problem of a point mobile robot using only tactile sensing (i.e. the sensors only indicate if the mobile robot is in contact with an obstacle). The algorithm is later adopted for robots with vision sensing systems (Lumelsky and Skewis, 1988). Their algorithms are shown to have strong convergence properties, but so far, they have been only used in static environments.

Feng and Krogh (1990) present a feedback approach to the local navigation problem where only local information is available. In their method, the problem of driving a robot to a goal in an unknown environment is formulated as a dynamic feedback control problem. The local feedback information is used to make steering decisions dynamically while the robot is moving. The developed strategy consists of two parts: subgoal selection and

steering decision. The subgoal selection algorithm generates temporary subgoals to be pursued when the final goal is not in sight. The steering decision algorithm chooses reasonable steering vectors using simplified representations of the robot dynamic and kinematic constraints to drive the robot toward the current subgoal. The algorithms guarantee the avoidance of the obstacles under the robot dynamic and kinematic constraints, and convergence to goal. Again, the algorithms are only used for the local navigation in static unknown environments.

2.1.4 Avoidance of moving objects with unknown trajectories

In subsection 2.1.2 we reviewed the work for moving robot among moving obstacles with known trajectories. All that work assume that full information about the description of the objects and their motion is available. However, in realistic environments, the information is based on measurements by a sensing system and the motion of the objects is not known in advance. Clearly the methods developed for the cases where the trajectories of objects are known are ill-suited to the problem which involves moving objects with unknown motions. If we apply those methods, we could plan a path initially, but the path would become invalid when an object changes its speed or trajectory.

One method of handling the moving objects with unknown motions or trajectories is referred to as incremental planning. With this method, we plan a path as if we have known the trajectories of the objects. This requires us to make a prediction. The prediction we make may or may not be correct. However, if the prediction is good enough so that we can use it for at least some minimum time period, we will make some progress toward the goal. We use the prediction until it becomes invalid, and then make a new prediction, and replan.

The navigation strategy presented in this thesis is similar in spirit to the incremental planning scheme in that we make a decision based on the prediction of the objects' motion. However, in our method, the time period for a valid prediction is not such an important factor as that in the incremental planning scheme. This is because the decision-making in our method is a local and real-time process, and the prediction is to be done frequently (i.e. at each sampling time) which allows moving objects to move in an uncertain way.

The incremental planning scheme has been used by several researchers. Slack and Miller (1987) describe a system which is capable of using this scheme to handle the unpredictability of the moving objects. In their approach, the workspace is divided into discrete cells, using a grid. This is implemented by using an array of processors. Each cell is represented as a processor, and adjacent processors are connected and can pass messages back and forth. Each message is associated with an energy value that reflects the cost of moving the robot to the adjoining 'node'. Associated with each cell is an emptiness score which is determined by a function much as Khatib's potential field. A relaxation algorithm is used to spread out the emptiness score. The path through the maximally empty cells can then be constructed. When the robot travels in a dynamic environment, it traverses one cell each time and then replan. This would require updating a few cells, and locally respreading the emptiness score. This approach suffers from the same problems as that of Khatib's potential field method.

Gil de Lamadrid and Gini (1990) also use the incremental planning scheme in their system. In their work, they try to drive a mobile robot to follow a predefined path among objects moving with unknown trajectories. The method they proposed interleaves path planning with execution. In their system, objects and the robot are represented by non-intersecting discs in the

plane and a point respectively. A tentative path is planned using predictions of the objects' movements in the future. The prediction is performed using whatever knowledge is available at planning time. When no knowledge is available, they assume that objects move in a straight line at a constant velocity. So they use the previous and current position of every object to predict its speed and line of motion. Since the path of the robot is predefined, they can compute the first predicted collision point, and choose a breakpoint from the current position to the goal that avoids that collision. This process is repeated recursively until a complete collision-free path is generated. The plan is executed as soon as it has been completely generated. While following the planned path, the robot relies on sensors to track the actual movements of the objects. Whenever the actual movements deviate excessively from the predictions, a new path is planned using updated prediction. Their experimental results show that their system handles unknown trajectories of the moving objects quite well. However, problems arise when complex instances (i.e. more than one moving obstacle) are presented. The complex instances require a great deal of planning time. On the other hand, the more moving objects are introduced, the more uncertain trajectories are produced which increase the probability that a plan will become invalid rapidly, especially when the moving objects are not moving in a straight line. Therefore, this method is not suitable to the problem in the presence of unpredictable moving objects which we address in this thesis.

Kant and Zucker (1988) propose to extend their method which we described in subsection 2.1.2, to deal with moving objects with uncertain trajectories. They describe a system where the path/velocity decomposition method for moving obstacle environment would be used to do global planning, and Khatib's potential-field method would be applied for local planning. This allows a limited amount of uncertainty in the position of the robot and obstacles, but in the velocities of the obstacles.

Kyriakopoulos and Saridis (1990, 1992, 1993) take Kant and Zucker's path/velocity decomposition approach further. In their system, objects, including the robot, are modelled as convex polyhedra, and a nominal trajectory is assumed to be known from off-line planning. A local decision-making strategy is then developed to control the velocity of the robot along the nominal trajectory to avoid collision with moving obstacles. They regard the distance estimation problem as a filtering problem. A Kalman filter is used to deal with uncertainty which is introduced by sensing and unknown dynamics of the moving obstacles. The possible collisions are then predicted based on the distance between the robot and a moving obstacle. Furthermore, three real-time collision avoidance strategies—optimal control strategy, minimum interference strategy and potential fields strategy, are developed and compared. Their simulation results show that the proposed scheme integrating collision prediction and a feedback type of avoidance for mobile robots in dynamic environments is computationally efficient and appropriate for real-time implementation.

However, this approach has some problems because the avoidance of moving objects is taken only by adjusting the motion along the predefined path. It is sometimes considered to be too restrictive since the robot is only allowed to move along the predetermined path to avoid collisions. This method can handle certain amount of uncertainty in both positions and velocities of the moving obstacles in real-time. However, it still may not give a solution in the case where the robot and a moving object approach each other.

Recent efforts to develop intelligent autonomous robotic agents capable of interacting with a dynamic environment have led to a variety of alternatives to traditional planning method. Brooks (1986) proposes a layered architecture for robots where activities such as obstacle avoidance are achieved by reacting

to the environment. Brooks' work is aimed at obtaining purposeful actions without any use of plans (Brooks, 1991). All actions are determined by the output of multiple concurrent, independent decision-making processes called *behaviours*. Each behaviour receives sensory input which is processed to suit its particular decision-making needs. In this way, the system is capable of responding to its environment in real-time without the delays imposed by sensor fusion.

Although the focus of our research is not on architecture-construction, Brooks' idea of emphasising a more direct coupling of sensing and action to make a robot achieve real-time activities is very useful. We adopt this idea in our project to find a solution to the problem of handling uncertain motions of moving objects in the area of motion planning.

In this section, we have discussed previous work related to the problem of motion planning. These demonstrate that the problem of avoidance of stationary known objects is computationally difficult. Problems with moving objects and unknown trajectories are even more difficult, and navigation in the presence of uncertainties in the obstacles' motion is still replete with open problems.

In the following sections, we introduce several key concepts and features of neural networks. This should provide a background for the subsequent description of our work.

2.2 Neural Network

As stated in the last section, the studies of robot motion planning is far from complete and the search for new computational techniques is a rather natural step in accomplishing autonomous robot performance. In this section, we

introduce a new computational architecture — the neural network, which is an effective way of achieving the autonomous navigation strategy that we proposed.

2.2.1 Introduction to neural networks

The study of neural networks started some fifty years ago in an effort to understand the enormous computational properties of the human brain. It was believed that a large collection of neurons, which may be modelled as simple nonlinear processing elements, are responsible for functioning of the brain. Artificially simulating or implementing this structure has given birth to what is now called (artificial) neural networks.

The neural network models go by many names such as connectionist models, parallel distributed processing models, and neurocomputing systems. Whatever the name, all these models attempt to achieve good performance via dense interconnection of simple computational elements. In this respect, the neural network structure is analogous our present understanding of biological nervous systems. Neural network models therefore offer greatest potential in areas such as vision, speech recognition, and robotics where many hypotheses are pursued in parallel, high computation rates are required, and the current best systems are far from equalling human performance.

Instead of performing a program of instructions sequentially as in a von Neumann computer, neural network models explore many competing hypotheses simultaneously using massively parallel nets composed of many computational elements connected by links with variable weights. They have the potential to be very fast and they are robust in the face of noise or incomplete data. In some cases, networks can be trained to solve a problem for which a specific formula or algorithm does not exist. Most neural net

algorithms also adapt connection weights in time to improve performance based on current results.

Among various features of neural networks, adaptation learning is the most attractive one. Neural networks can learn how to perform certain tasks by undergoing training with illustrative examples. While neural networks learn to discriminate patterns, we do no have to build elaborate *a priori* models, nor do we need to specify probability distribution functions. The designer's sole concern is designing appropriate architecture and developing a good *learning algorithm* which would enable the network to learn to discriminate from a set of *training patterns*. Furthermore, after training, the network can subsequently not only recognise such patterns when they occur again, but also recognise similar patterns by generalisation. The special features of adaptation learning and generalisation provide neural networks the capability of handling unexpected events or inputs.

In the following subsection, we describe several typical models of neural networks and introduce their characteristics.

2.2.2 Neural network models

Neural networks are specified with the net topology, node characteristics, and training or learning rules. These rules specify an initial set of weights and indicate how weights should be adapted during use to improve performance. In this subsection, we introduce three most widely used models of neural nets, and provide some insights concerning the principles of neural nets operation.

Neural network topologies or architectures are formed by organising **Processing Elements** (PEs) into layers and linking them with weighted interconnections. PEs are also referred to as nodes, neurons, or threshold logic

units. The basic PEs employed in neural networks differ in terms of the type of network considered. However one commonly encountered model is a form of the McCulloch and Pitts neuron (McCulloch and Pitts, 1943) as shown in Figure 2-1. The input signals come from either the environment or outputs of other PEs. Associated with each connected pair of PEs is an adjustable value called a weight. The output of the node is found as a function of the summed weighted strength inputs. The node is characterised by the nonlinearity of the activation function or threshold function f() and the internal threshold value or bias term[1]. For nonlinear transfer functions, many different nonlinearities are possible, but the two most commonly used ones are the hard limiters, and sigmoidal nonlinearities.

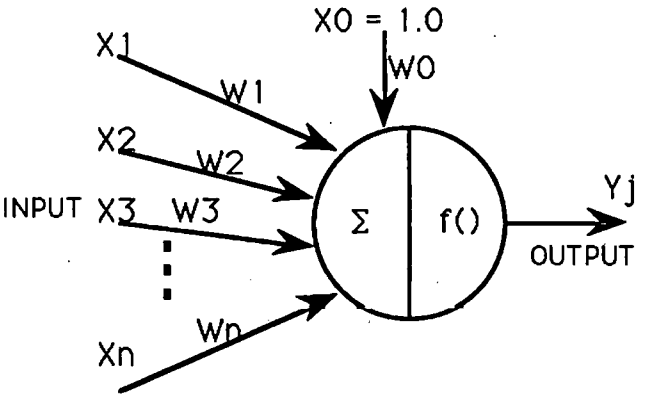


Figure 2-1. Processing element

a) Multilayer Feedforward Network

The multilayer feedforward network[2] model was originally introduced by Rumelhart and McClelland (1986). As shown in Figure 2-2, a multilayer

---

[1] This bias term of a unit can be thought of as the weight of an additional link coming from a hypothetical unit which is always on (i.e. with value equal to 1). We thus use the word "weight" to refer to the connection weight between two units as well as the bias of a unit.

[2] This structure is sometimes referred to as multilayer perceptron as an extension of Rosenblatt's definition for perceptrons, or referred as backpropagation network due to the learning algorithm.

feedforward network has one input layer, one output layer, and at least one hidden layer. The units in the different layers are often given different names, such as input units, hidden units and output units. Each layer is fully connected to the succeeding layer.

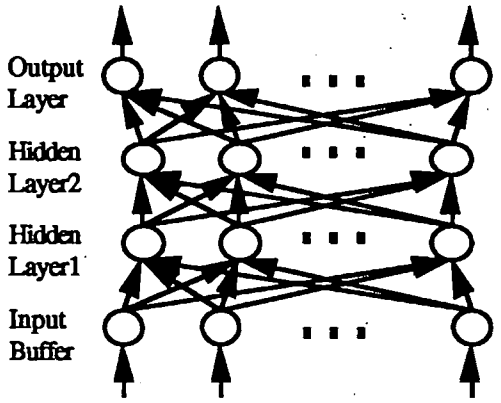


Figure 2-2. A Multilayer feedforward network

The *input units* basically receive inputs from sources external to the network. These inputs may be sensory inputs from the environment in which the system is embedded. *Hidden units* are not directly connected to the external world, therefore they are not visible to outside of the network. *Output units* send signals out of the network. During recall, the input that appears at the input layer is propagated to the second layer, then the output of the second layer is propagated to the third layer, if there is one, and this goes on until the final layer is reached.

An output function for each processing unit: As can be seen in Figure 2-1, the outputs of units in layer k transfers its inputs as follows:

$$x_j[k] = f(S_j[k]) = f\left(\sum_i (w_{ji}[k] * x_i[k-1])\right) \qquad (2.1)$$

$x_j[k]$: Current output state of j^{th} unit in layer k.

$w_{ji}[k]$: weight on connection joining i^{th} unit in layer (k-1) to j^{th} unit in layer k.

$S_j[k]$: weighted summation of inputs to j^{th} unit in layer k.

Here, we use a superscript in square brackets to indicate which layer of the network is being considered, and f() can be any differentiable monotonic function but a common choice is the sigmoid function defined as:

$$f(z) = (1.0 + e^{-z})^{-1} \tag{2.2}$$

Backpropagation learning algorithm: Given a set of input-output data, the learning task is to adjust the network weights such that the desired mapping between the input and the output can be captured over a subset of the input space defined by the training data. A weight update algorithm for this purpose is derived in Rumelhart and McClelland (1986) and referred to as the *backpropagation algorithm* due to its structural properties.

The algorithm is based on the idea of minimising a cost function obtained as the summation of the squared error terms of the network's output layer, defined by:

$$E_p = \frac{1}{2} \sum_i (d_i - o_i)^2 \tag{2.3}$$

where the index i refers to the individual units in the output layer and p is the index for the sample patterns. $(d_i - o_i)$ is the error term for each individual output unit with d_i and o_i representing the desired and actual output values, respectively. Note that the cost function given by (2.3) is defined for one sample pattern. Minimising this cost function over the weight space of the network forms the basis of the backpropagation algorithm. A simple gradient descent rule can be utilised on (2.3) to obtain the weight update equation as,

$$\Delta w_{ji}^{[k]} = - l_{coef} * (\partial E_p / \partial w_{ji}^{[k]}) \tag{2.4}$$

where l_{coef} is a learning coefficient, the gradient term ($\partial E_p / \partial w_{ji}^{[k]}$) in (2.4) has to be evaluated at each layer of the network in order to obtain the update equations for all the network weights.

Replacing the above gradient term as in Rumelhart and McClelland (1986), the complete weight update equations can be derived as follows:

$$\Delta w_{ji}^{[k]} = - l_{coef} * e_j^{[k]} * x_i^{[k-1]} \qquad (2.5)$$

$$e_j^{[k]} = d_j^{[k]} - o_j^{[k]} , \qquad \text{for output layer} \qquad (2.6)$$

$$e_j^{[k]} = f'(S_j^{[k]}) * \sum_i (e_i^{[k+1]} * w_{ij}^{[k+1]}) , \qquad \text{for any hidden layer} \qquad (2.7)$$

where $e_j^{[k]}$ is considered a measure of the local error at processing element j in level k. An explicit derivation of this algorithm is given in Rumelhart and McClelland (1986).

The weight update equations that are defined above are generally simulated in a modified form in order to achieve better convergence properties. They are modified in the following form,

$$w_{ji}^{[k]}(t) = - l_{coef} * e_j^{[k]}(t) * x_i^{[k-1]}(t) + \mu * \Delta w_{ji}^{[k]}(t-1) \qquad (2.8)$$

where t denotes the iteration number in the update process; $\Delta w_{ji}^{[k]}(t-1)$ is referred to as the momentum term and μ is the momentum coefficient which determines the effect of past weight changes on the current direction of the gradient in the weight space. This provides a kind of momentum in weight space that effectively filters out possible high-frequency variations of the error surface. This modification is originally proposed in Rumelhart and McClelland (1986).

Note that the update equations constitute a backward recursion. They are repeated for a set of sample patterns until the total squared error given by (2.3)

drops below a predetermined level for each pattern[3]. The weights obtained through this training procedure are then used in generating the desired input-output mapping.

Nonlinear function approximation: The network structure illustrated in Figure 2-2 basically defines a mapping between input and output vector spaces. There has been experimental evidence that the mapping of the multilayer feedforward network is capable of approximating arbitrary nonlinear functions of its input space (Lapedes and Farber, 1987; Gorman and Sejnowski, 1989). A number of research papers have also given mathematical proofs showing that multilayer feedforward network can be used as functional approximators (Funahashi, 1989; Hornik and White, 1989). The proofs make this type of network a much stronger candidate compared to other types of network for many applications. Various multilayer feedforward networks have been successfully applied to solve problems in wide areas, particularly, in pattern recognition, signal processing and control applications (Werbos, 1989) although there is no known method for determining the number of nodes and layers, convergence parameters, etc., for a given problem.

b) Hopfield network and Kohonen network

Hopfield network: One of the major contributions to the area of neural networks was made in the early 1980's by John Hopfield (1982). Unlike the multilayer feedforward networks, the Hopfield network is a single-layer network with feedback connections from the outputs of the network to its inputs as shown in Figure 2-3. Each node uses a hard-limiting nonlinearity (variations use other nonlinearities). The Hopfield network processing element transfer function is given by

---

[3] Note that this error level determines the termination of the training process for off-line applications.

$$x_i^{new} = \begin{cases} 1 & \text{if } \sum_{j=1}^{n} w_{ij} x_j^{old} > T_i \\ x_i^{old} & \text{if } \sum_{j=1}^{n} w_{ij} x_j^{old} = T_i \\ -1 & \text{if } \sum_{j=1}^{n} w_{ij} x_j^{old} < T_i \end{cases} \qquad (2.9)$$

for i = 1, 2, ..., n. The processing elements of the network updated one at a time. The only constraint on the scheduling of these updates is that all of the processing elements must be updated at the same average rate. The values T_1, T_2,..., T_n are thresholds.

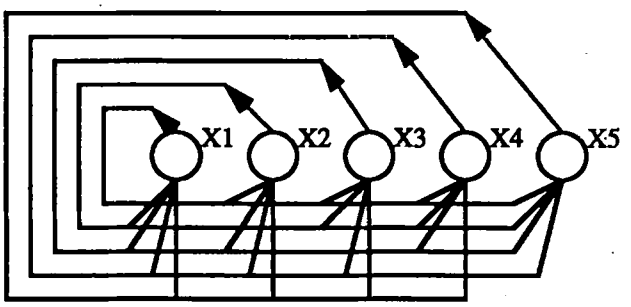


Figure 2-3. A Hopfield network

The Hopfield network does not have a learning law associated with its transfer function. The n x n weight matrix W = (w_{ij}) (which is therefore fixed) is assumed to be specified in advance. No restrictions on the real number values w_{ij} are made, except that the matrix W must be symmetric and have a 0 diagonal ($w_{ij} = w_{ji}$ and $w_{ii} = 0$).

In the Hopfield model, inputs to the network are applied to all nodes at once, and consist of a set of starting values, +1 or -1. The network is then left alone, and it proceeds to cycle through a succession of states, until it converges on a stable solution, which happens when the values of the nodes no longer alter. The output of the network is taken to be the value of all the nodes when the network has reached a stable, steady state.

The behaviour of the Hopfield network can be characterised by means of an energy function. The idea is that the function

$$E(x) = -\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\, x_i\, x_j + 2\sum_{i=1}^{n} T_i\, x_i \qquad (2.10)$$

always decreases whenever the state of any node changes. In other words, the output update serves to perform a minimisation of the energy function. This property can be used for solving combinatorial optimisation problems (Wilson and Pawley, 1988). An example is the travelling salesman problem in which a salesman must visit N cities and minimise the distance travelled over the entire tour. The only information given is the distance between pairs of cities. To solve this problem an energy function is defined so that the minimisation of this function corresponds to a solution of the problem.

An advantage of the Hopfield network is that there exists a proof of convergence based on Lyapunov analysis (Wilson and Pawley, 1988) and, furthermore, the convergence rate is independent of the number of nodes. However, even though the network is guaranteed to converge, it often converges to local minima which are undesirable, e.g. the salesman does not visit every city. In the travelling salesman problem the energy function can be easily defined since the solution only needs to meet a limited number of constraints. In more difficult problem with large numbers of constraints it seems that the network will be even more likely to arrive at invalid solutions.

Kohonen network: In many ways Kohonen (1984) followed on directly from Hopfield by using a single-layer network as a basis for speech recognition. Kohonen network, also referred as the self-organising feature map, consists of a two-dimensional array of neurons. As shown in Figure 2-4, the neurons are not arranged in layers as in the multilayer feedforward network (input, hidden, output) but rather on a flat grid. All inputs connect to every node in

the network. Feedback is restricted to layeral interconnections to immediate neighbouring nodes. There is no separate output layer – each of the nodes in the grid is itself an output node.

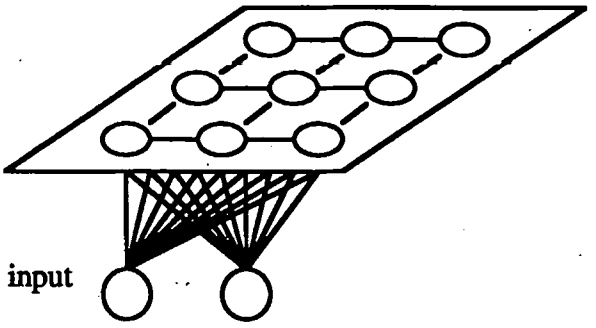


Figure 2-4. A Kohonen network

Without the *supervised learning* where the desired response of the network always being available for each input from the training patterns as backpropagation learning algorithm, the network organises itself using winner-take-all mechanism (or competitive learning) to form its own feature classifications on the training data. The learning law is defined as:

$$w_i^{new} = w_i^{old} + \alpha * (x - w_i^{old}) * z_i \qquad (2.11)$$

where x is a input vector and z_i is the output of the i^{th} node. α is a constant, $0<\alpha<=1$. Only the winning node actually gets to modify its weight. This is because the second term of the right-hand side of this equation is multiplied by z_i, which is set to 1 if it is the output of the winning node, otherwise is 0.

The two key factors to adaptive self-organising learning in a Kohonen network are the weight adaptation process and the concept of topological neighbourhoods of nodes. Both of these ideas are very different from the neural networks we discussed so far.

The learning process in a Kohonen network is described as follows. Initially all the connections from the inputs to the nodes are assigned small random weight values. Each node thus has a unique weight vector, the dimensionality of which is defined by the number of components in the input vector. During learning, each node measures the Euclidean distance of its weight vector to the incoming input vector. The node with the weight vector closest to the input pattern is selected as the "winner", and it adjusts its weights to be closer to the values of the input data. The nodes in the neighbourhood of the winning node also adjust their weights to be closer to the same input data vector. As a consequence, input vectors that are close spatially to the training values will still be classified correctly even though the network has not seen them before. This is one of the most useful and intriguing aspects of the Kohonen network. However, one problem that could arise is that, by chance, one node can end up representing too much of the input data. One of the proposed solutions is to produce a set of equiprobable weight vectors with respect to input data vectors in accordance with the probability density function (Desieno, 1988). For many types of data where the time sequence of presentation is unimportant, the probability density function is essentially everything that can be known about the data. Therefore, the ability of a Kohonen net to form an accurate, yet compact, representational model for probability density function of the training data can be of great value for problems in pattern recognition, statistical data analysis, control, and knowledge processing.

Before completing the introduction of neural networks, we should note that apart from the three models we introduced there are also many other models such as: ART, Associative memory and Neocognitron etc. The further introduction can be found in Wasserman (1989).

2.3 Summary

In this chapter, we discussed the related work in the area of motion planning and introduce several key concepts of neural networks.

To summarise, the state-of-the-art research provides a wealth of solutions to the mobile robot motion planning problem. Given complete information for the environment and sufficient time for computation, one can find a path to minimise desired cost functions. Given only local feedback information one can generate a sequence of points for the mobile robot to follow which would ultimately lead it to the goal in complex environments.

Yet none of these approaches is suitable for the dynamic navigation of mobile robots in realistic environments where exist unpredictable events which involve unknown stationary obstacles and unforeseen moving obstacles disrupting the motion of the mobile robot. The navigation/steering decisions must be made dynamically based on the sensor readings and the predictions of obstacles' motion as the robot moves toward its goal. In this context, the neural network techniques seem to be a powerful tool to be used in developing an autonomous navigation strategy under environmental uncertainty. This will be further discussed in the subsequent chapters.

Apart from the multilayer feedforward networks, there have been a number of other neural networks (for example, Hopfield net and Kohonen net), each of which has different strengths particular to their applications. However, the architectures of the multilayer feedforward networks are fairly well understood compared with other types of network, and procedures for applying them successfully in a number of application domains have been worked out. Therefore we focus on this type of network and use it to achieve

the real-time intelligent navigation strategy by means of three components (Planner, Navigator and Motion Predictor).

Throughout this research the number of layers in the multilayer network is always 3 (1 hidden layer), as this is theoretically sufficient for any input/output mapping, as described in section 2.2.2. Furthermore, the number of units in the hidden layer is always determined empirically by finding the minimum number of units that still allow the network to converge during training.

Chapter 3

A Neural Network Path Planner

3.1 Introduction

In chapter 1, we said that our navigation strategy consists of three components: planner, navigator and motion predictor. The **planner** is first called to generate a sequence of via points (subgoals) from the initial position to the goal position based on the map of the workspace. Once the nominal path is found, if a moving object is classified as disruptive, the **predictor** starts tracking the moving obstacle by predicting the trajectory of the moving object at the next sampling instant. Meanwhile, the **navigator** generates steering commands to drive the robot to each subgoal while avoiding collisions based on the sensor readings and the predictions of the moving objects.

In this chapter, attention is focused on the neural network path planning algorithm for generating a collision-free path from the initial position to the goal position based on the map of the workspace. The planned path consists of a sequence of via points (subgoals) which provide temporary directions that lead the robot toward the final goal. The objective of the proposed planner is to allow the robot to make high-level, goal-oriented decisions and also to be capable of achieving fast replanning once the navigator (presented in chapter 4 and 6) is unable to handle special situations during navigation. Ideally this should be real-time, but it is accepted that even using neural networks, dedicated computing would be required to achieve real-time re-planning.

With the assumptions made in section 1.2 of chapter one, the configuration

space is considered as a two-dimensional plane which is divided into equal size cells and stored in a map array. The robot is represented by a point and a set of obstacles are represented by a set of convex polygons. The path planning problem can thus be simplified as finding a sequence of collision-free via points among a set of polygons from the initial position to the goal position. As specified in subsection 2.1.1 of chapter 2, many algorithms have been developed for this problem. However, existing schemes require some preprocessing and graph searching, and thus are generally computationally intensive and cannot be implemented on-line. The neural network approach appears to be very promising for accomplishing real-time collision-free path planning with its powerful parallel computation and distributed representation.

In the following sections, we first briefly review previous research using neural network techniques to tackle the path planning problem. This is to provide a context for discussion of the planning algorithm that we developed. We then describe our algorithm in detail and finally discuss the test results.

3.2 Neural Computation for Path Planning

In order to find the global optimal path, a constrained optimization formulation can be adopted. The key is to define a cost function using several important variables, e.g., the distance between the initial position and the goal position, etc. A path can thus be found by minimizing the cost function. As specified in subsection 2.2.2 of chapter 2, neural networks can be applied to the optimization of cost functions.

Jorgenson (1987) has investigated the use of *simulated annealing* (Rumelhart and McClelland, 1986) for mobile robot path planning. In his approach, a workspace is represented as a lattice of connected voxels (3-dimensional pixels)

formed by dividing the workspace into equal sized cells. Each voxel is associated with a continuous valued Hopfield's node. The magnitude of a node activation corresponds to a probability of voxel occupancy. The planning process merges a nearest neighbour grid cell technique and a simulated annealing gradient descent method to optimize traversal movements. However, Jorgenson noted that simulated annealing is too computationally intensive for real-time applications.

Tsutsumi et al (1988) applied Hopfield nets to find a path for multi-joint manipulators and truss structure in an unknown environment. In their system, the manipulator is modelled by using the distances between the joint locations instead of the polar coordinate. The relationship between the environment and the joints is also represented by their mutual distance. To guide the manipulator to the target position without any collisions, the constraints of the operation based on the distances are defined by using energy functions. The analog Hopfield network is employed to actualize parallel control by minimizing the total of the energy functions based on the distances.

In Tsutsumi's later work (1989), a multi-layered network composed of Hopfield and Backpropagation nets is proposed in consideration of the avoidance of the deadlocking state. In the network, a Hopfield net is used for the energy minimisation and two Backpropagation nets put in the front and in the rear of the Hopfield net function as mapping networks with a learning capability. The input signals from the environment are mapped via one Backpropagation net into the internal space, where the Hopfield net minimizes the total energy according to the internal space representation. The output signals of the Hopfield net are mapped again to the environment via the other Backpropagation net with the inverse mapping. The simulation studies show that the proposed network helps the end of the manipulator to reach the target point through the shortest pass in the internal space. However,

the energy function used to describe the network weights in the Hopfield net is quite complex and thus, it is not clear if this approach is feasible to implement for mobile robots in practice.

Seshadri (1988) also used Hopfield nets for mobile robot path planning. His formulation resembles the travelling salesman problem in that the purpose of the neural network is to minimize the length of a path to a goal position. However, no details of the energy functions are given.

The method used in our planner originated from that suggested by Park and Lee (1990). The real-time path planning algorithm they developed is based on multilayer feedforward type networks. They approached the path planning problem by utilizing Hopfield's neural network optimization concept (Hopfield and Tank 1986). A path is represented by a set of via points, its collision penalty thus can be considered as a sum of individual collision penalties of all the via points. The collision penalty function is quantified by a three-layer neural network for each obstacle as shown in Figure 3-1.

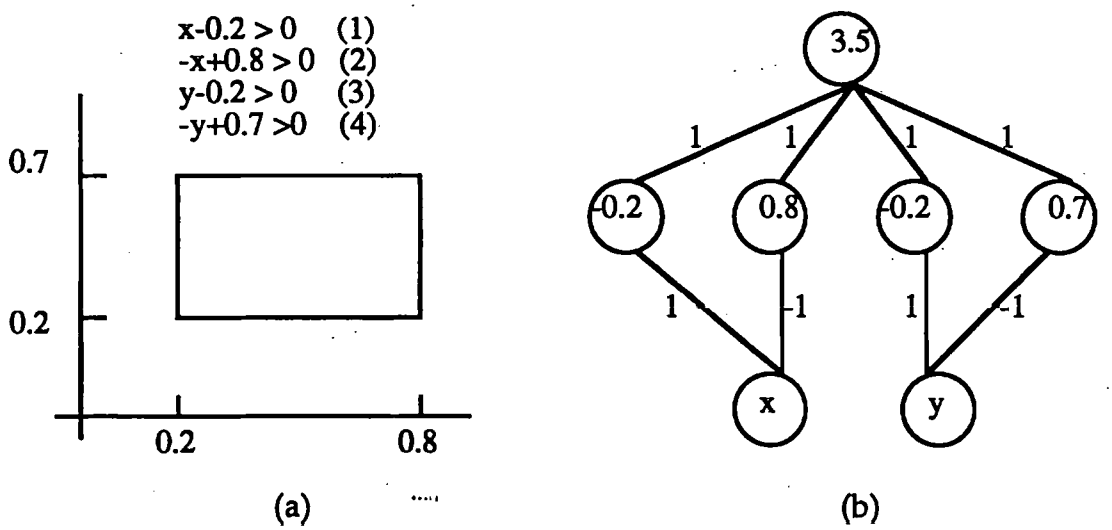


Figure 3-1. An example of the neural network collision penalty for a rectangular obstacle: (a) The obstacle and the mathematical inequality constraints. (b) Network representation.

An obstacle is defined by a set of inequalities which are mapped one-to-one to edges of the obstacle, as shown in Figure 3-1(a). When the coordinates of a point are substituted, if all of the inequalities are satisfied then that point is inside the obstacle. The set of inequalities can be mapped directly onto a 3-layered neural network as shown in Figure 3-1(b). In the network, each of the bottom layer units represents respectively the x, y coordinate of a via point. The connections between the bottom layer and the middle layer are assigned to the coefficients of x, y in the inequalities, and the threshold of a middle layer unit is assigned to the constant term in its inequality. The connections between the top layer and the middle layer are all assigned to 1, and the threshold of the top layer unit is assigned to 0.5 less than the number of inequalities. The output of the network is 1 when the coordinates correspond to a point within the obstacle, and 0 otherwise. This output is the collision penalty associated with an obstacle. With a number of obstacles, the same number of networks are needed i.e. if there are N objects then there are N separate networks.

Note that the penalty function associated with an obstacle is based on the shape and the position of the obstacle. The inequalities which represent an obstacle must be formulated according to the shape and the position of the obstacle, and assigned to the network in advance. However, if there are many obstacles in the workspace, the inequalities which describe each obstacle must be calculated in turn and assigned to different networks. In addition, it would be computationally more expensive in situations where the obstacles with complex shapes exist. Because of these problems, this algorithm is only suited to off-line path planning and cannot be directly used by our planner (which should be able to replan the path on-line and in real-time).

In the next section, we describe our proposed real-time path planning algorithm which is an improvement on Park and Lee's algorithm.

3.3 An Enhanced Path Planning Algorithm

In our enhanced method, the collision penalty function associated with all the obstacles in the workspace is derived when a backpropagation network has completely learnt the relationship between the coordinate of the workspace and its corresponding occupation state. This is different from Park and Lee's algorithm we described above, where the collision penalty generated by each obstacle has to be one-by-one assigned to different networks.

Given the representation of the collision penalty, the path planning procedure in our algorithm is as follows:

Given the coordinates of the initial and goal points, via points of the path are arbitrarily chosen between the initial point and the goal point. These via points are desired to move in space, and to converge eventually to the positions which make the optimal collision-free path.

In the following subsections, we first present the architecture of the backpropagation network for determining the collision penalty function. By defining the energy and then deriving the dynamical equations, the path planning algorithm is then developed for a point mobile robot. Finally, a cooling schedule technique taken from simulated annealing is utilized to improve the performance of this method.

3.3.1 Collision penalty function–Neural network representation

In the proposed approach for real-time path planning, a path is described by a set of via points. In this way, we can localize a path planning problem at individual via points, allowing massive parallel and distributed computing.

To quantify the collision between a path and obstacles, the collision penalty of a path is defined as the sum of individual collision penalties of all the via points, where the collision penalty of a via point is obtained from a network representation. We use a single hidden layer backpropagation network shown in Figure 3-2 with a set of arbitrary small weights to learn the representation of the collision penalty function. The sigmoid function

$$f(x) = \frac{1}{1 + e^{-x/T}} \tag{3.1}$$

is used as the activation function at each unit in the network. Note that T is the measure of 'temperature' and it can play an important role in improving the performance of the algorithm as will be discussed in subsection 3.3.3.

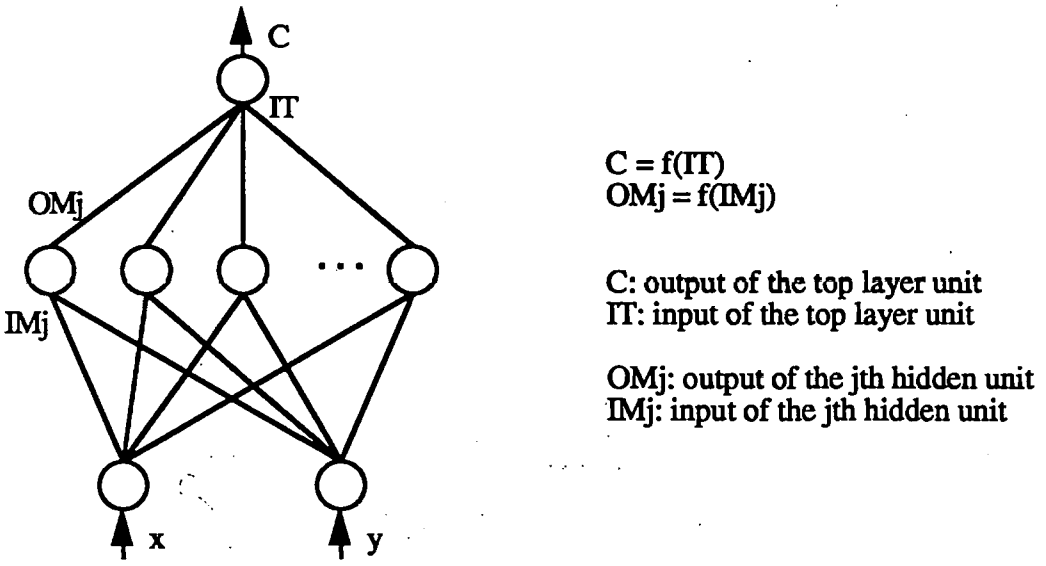


Figure 3-2. Network for collision penalty function

In the learning process, the training patterns are collected from the workspace which is divided into 2-D grid cells. A binary value is assigned to each grid cell to indicate the clearness status. Namely, a value '1' implies that the cell is fully occupied, while '0' implies that the cell is unoccupied. In this sense, each obstacle in the workspace is represented by a array of occupied cells. The training patterns representing the workspace can thus be constructed as pairs of

the coordinate of a cell and its corresponding occupation state, and then presented to the backpropagation network. When the network with formed connection weights converges to the desired patterns its learning process is completed. The collision penalty representation is thus derived by the association between the x, y coordinate of each point in the workspace and its occupation state.

In the trained backpropagation network, if the x, y coordinate of a via point is given to the bottom layer units, the top layer unit outputs the continuous value from 0 to 1 according to the degree that a given point collides with the obstacles. Thus, a path collides with the obstacles in the workspace if any of its via points forces such a network to output the value which approximately equals 1.

3.3.2 Energy and dynamical equations

A collision-free path planning problem is equivalent to the optimization problem with two constraints. One is to avoid colliding with obstacles, and the other is to minimize the length of the path. These two factors are to be considered in defining the energy function. The path planning algorithm developed here is based on minimizing the energy in terms of the total path length and the collision penalty.

With the representation of the collision penalty of a via point in the subsection above, the collision penalty of a path is defined as the sum of the collision penalties of all the via points. We thus define the energy for collision as:

$$E_C = \Sigma_{i=1,N} C_i \qquad (3.2)$$

where N is the number of via points, and C_i is the collision penalty at the i^{th} via point, P_i.

As for the path length, the energy is simply defined as the sum of squares of all the lengths of the line segments connecting via points, $P_i(x_i, y_i)$, for i = 1, 2, ..., N:

$$E_L = \Sigma_{i=1,N} L_i^2 = \Sigma_{i=1,N} [(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2] \quad (3.3)$$

where L_i is the i^{th} line segment length. As the path converges to the minimal one, E gets smaller, and the via points tend to be equally separated due to the usage of its square rather than the line segment length itself. With those two measures, we define the total energy, E, as:

$$E = E_L + E_C \quad (3.4)$$

Since low energy implies less collision and a short path, the dynamical equation for a via point is chosen to make the time derivative of the energy negative along the trajectory. The derivative of E with respect to time is

$$\begin{aligned} dE/dt &= \Sigma_i (\nabla P_i E)\, dP_i/dt \\ &= \Sigma_i \{ [\partial L_i/\partial x_i + \partial C_i/\partial x_i] * dx_i/dt + [\partial L_i/\partial y_i + \partial C_i/\partial y_i] * dy_i/dt \} \end{aligned} \quad (3.5)$$

By choosing the time derivative of each coordinate as:

$$\begin{aligned} dx_i/dt &= -[\partial L_i/\partial x_i + \partial C_i/\partial x_i] \\ &\cdots \\ dy_i/dt &= -[\partial L_i/\partial y_i + \partial C_i/\partial y_i] \end{aligned} \quad (3.6)$$

the time derivative of energy along the trajectory becomes:

$$dE/dt = -\Sigma_i \{ [dx_i/dt]^2 + [dy_i/dt]^2 \} < 0 \qquad (3.7)$$

$dE/dt = 0$ if and only if $dx_i/dt = 0$ and $dy_i/dt = 0$. This means that all via points move along trajectories that decrease the energy, and finally reach equilibrium positions. From Eq. (3.6) and Figure 3-2,

$$\begin{aligned} \partial C_i/\partial x_i &= \partial C_i/\partial IT * \partial IT/\partial x_i \\ &= f'(IT) * \Sigma_{j=1,J} \partial OM_j/\partial x_i \\ &= f'(IT) * \Sigma_{j=1,J} f'(IM_j) * w_{xj} \end{aligned} \qquad (3.8)$$

where C_i is the output of the network at i^{th} via point, IT is the input of the output layer unit, OM_j is the output of the j^{th} hidden layer unit, IM_j is the input of the j^{th} hidden layer unit, J is the number of hidden units. From Eq. (3.6) and (3.8) , the dynamical equations of the via point $P_i(x_i, y_i)$ is derived as :

$$\begin{aligned} dx_i/dt &= -[(2x_i - x_{i-1} - x_{i+1}) + f'(IT) * \Sigma_{j=1,J} f'(IM_j) * w_{xj}] \\ dy_i/dt &= -[(2y_i - y_{i-1} - y_{i+1}) + f'(IT) * \Sigma_{j=1,J} f'(IM_j) * w_{yj}] \end{aligned} \qquad (3.9)$$

where $f'()$ is equal to $\frac{1}{T} * f() * [1 - f()]$ with the sigmoid function Eq. (3.1). This analysis is similar to that of the back-propagation algorithm. The difference is that the input is varied in this algorithm, while the connection weights are adjusted in the back-propagation algorithm. Note that the algorithm can be implemented in parallel by using the same number of networks of this kind as that of the via points. Each via point is used as the input to its corresponding network. The parallelism involved in the computations is thus associated with via points.

3.3.3 The effect of temperature in the penalty function

A problem exists in the algorithm developed above, which is that the energy surface contains extensive flat areas with very little or no slope. The backpropagation method, being a gradient descent, gets stuck in these areas. These areas are created because the whole region inside the objects has a collision function equal to unity. So even if the via point moves, the total energy changes very little because the collision function is constant. This problem is partially overcome by the use of the sigmoid function in the output unit. This has the effect of smoothing the edges of the object so that when the via point is close to the edge of an object, a gradient exists which the backpropagation algorithm can follow.

The introduction of temperature into the sigmoid function increases the degree of the slope at the edges of the object. When the temperature is high, the slope extends further into the object, and thus eliminates the flat areas. Backpropagation moves the via points out of the object, so that as time goes on, the temperature can be lowered, effectively creating a barrier which prevents the via points from moving back into the objects.

Park and Lee call this process "simulated annealing" because of the use of a cooling schedule. However, it must be pointed out that simulated annealing involves more than just a cooling schedule, as it also employs probablistic units which allow jumps to higher energy states. This overcomes the problem of local minima, but in this application, where probablistic units are *not* used, it is impossible for the network to jump to a higher energy state. The process, which is just gradient descent, is therefore quite different from simulated annealing, but still improves the method described by allowing the network to move away from flat areas.

Using the relationship between the parameter T and the shape of the surface of the penalty function, the network is able to escape from flat areas by slowly decreasing T from the sufficiently high temperature.

Szu (1986) suggest an efficient schedule for simulated annealing which can be applied here ,

$$T(t) = \frac{T_0}{1 + t} \tag{3.10}$$

where T(t) is the current temperature, t is time, and T_0 is a sufficiently high value of the initial temperature. The fast simulated annealing schedule, Eq. (3.10), allows the path planning to be done with reasonable computation time.

3.4 Test Results

In this section, we examine the behaviour of the proposed algorithm through computer simulation. We first present the test example, and then discuss the strengths and the weaknesses of the algorithm.

3.4.1 Test example

The neural network based path planning method that we developed has been tested on a computer simulation which is performed in two steps. In the first step, a two dimensional workspace which consists of 8 by 8 grid cells as shown in Figure 3-3(a), is presented to a single hidden layer backpropagation network. As illustrated in Figure 3-2, the network consists of 2 input units, 1 output unit and the number of hidden units is empirically fixed to 25 by using a simulation package named "NeuralWorks Professional II/Plus and NeuralWorks Explorer", which runs on a Macintosh II. The number of hidden units is a result of a trade-off between two conflicting requirements:

1) to provide enough units so that the network can accomplish the input/output transformation;
2) to keep the number of hidden units as low as possible to achieve good generalisation (an excess number of hidden units would make the network act as a look-up table).

The purpose of training is to generate the collision penalty function associated with the obstacles in the workspace. The training task is accomplished by using the simulation package, where the generalized delta rule with a momentum term is chosen to adjust the connection weights. The training patterns are collected from Figure 3-3(a), where the inputs to the network are x, y coordinate values, and the desired outputs to the network are binary values which indicate the occupation state of the corresponding cell. During training, we initially set larger values to the momentum coefficient and learning rate to get fast convergence, which are respectively assigned to 0.7 and 0.5. After 5000 sweeps, the momentum coefficient and the learning rate are decreased to 0.35 and 0.15 respectively, which protects the training process from large oscillations. After 241658 sweeps[4] of training on the 64 training patterns, the network satisfactorily converges to the desired patterns with a small Root Mean Square (RMS) error 0.0049. The network which represents the collision penalty function associated with obstacles, is thus derived with connection weights as shown in Table 3-1. Note that, the RMS error is a common measure of the convergence degree for training a network, and it is calculated by adding up the squares of the errors for each training pattern, dividing by the number of patterns to obtain an average, and then taking the square root of that average. RMS error can also represent the error over many output units, if there are.

In the second step of the simulation, the trained network is used to find a path

[4] If the NeuralWorks Professional II simulation package is used, the "sweep" means updating the weights after every pair (input, desired output) presentation.

initial position is chosen at (0, 0), and (7, 7) for the goal position.

TABLE 3-1 Connection weights and bias terms

Hidden Units	Bias	Input x	Input y	Output Node
Bias				-4.1109
Node1	0.6258	-1.2008	-0.6996	0.2725
Node2	6.8406	-1.8441	-1.0154	3.6129
Node3	1.7354	-1.0861	-0.7384	0.5971
Node4	3.5914	-1.1338	-0.8901	1.5145
Node5	1.4074	-1.1294	-0.7344	0.4927
Node6	1.9276	-1.0692	-0.743	0.6838
Node7	7.1089	-4.9729	1.4948	-7.466
Node8	10.4327	-7. 8297	0.9498	-11.8768
Node9	27.7355	-7.2171	1.9812	9.1468
Node10	10.2783	-4.6651	0.9549	-6.7623
Node11	0.4733	-1.275	-0.8239	-1.4147
Node12	1.873	-1.0736	-0.742	0.6538
Node13	4.183	-1.1934	-0.9601	1.8689
Node14	0.7652	-1.2004	-0.7133	0.3181
Node15	1.773	-1.0811	-0.7392	0.622
Node16	8.8672	-2.4067	-1.0451	5.1978
Node17	1.2624	-1.1518	-0.7319	0.4329
Node18	10.0979	-2.7136	-1.12	6.2288
Node19	0.7371	-1.2009	-0.7108	0.3087
Node20	-0.0314	-2.8889	0.9803	-0.9112
Node21	1.6268	-1.0981	-0.7366	0.5642
Node22	2.3122	-2.3914	0.6327	-2.9606
Node23	9.2716	-2.5082	-1.0682	5.533
Node24	1.0112	-1.1826	-0.7274	0.3938
Node25	19.0511	-0.0125	-4.3657	-9.9122

During path finding, the initial path is arbitrarily chosen as a straight line from the initial position to the goal position as shown in Figure 3-3(b). The fast simulated annealing schedule (3.10) is applied to the path planning procedure to avoid flat areas. The parameter T_0 is set to 3.5, and the temperature is gradually decreased to 0.7 at time t=4. The time, t, is adjusted in steps of 0.5 units. Figure 3-3(c) illustrates the trajectories of via points during convergence, and Figure 3-3(d) shown the final collision-free path. Note that the network training part is performed using the simulation package and the rest of the simulation is implemented in Turbo-C on a PC-386SX.

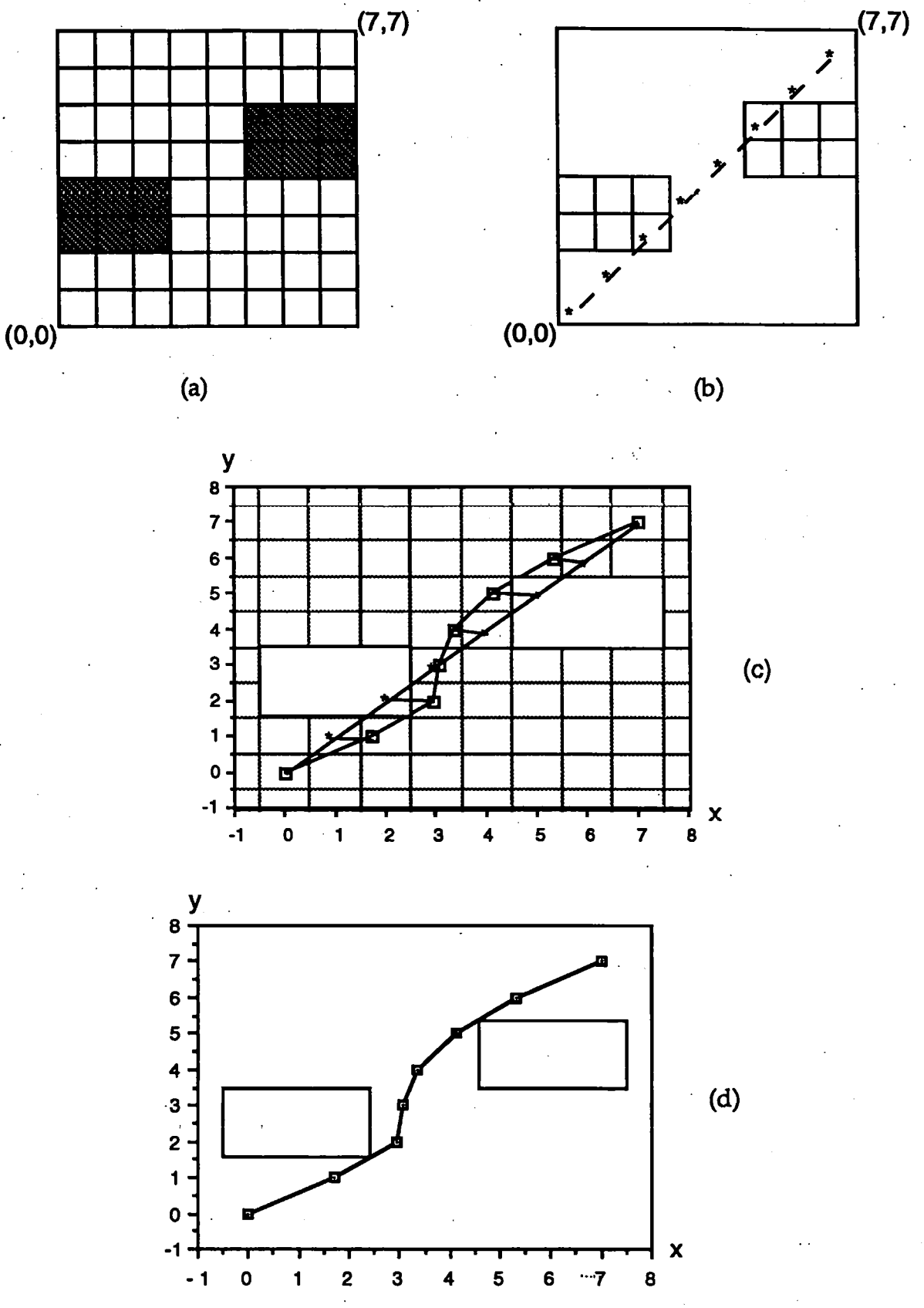


Figure 3-3. Path planning with two obstacles (a) Workspace representation (b) Initial path (c) Trajectories of the via points (d) Final path.

The remarkable aspect of this example is that it finds a collision-free path in only 8 iterations. Compare this to the thousands of iterations normally associated with backpropagation in neural networks. It is assumed that this is at least partially due to the effect of the temperature in the output function.

3.4.2 Discussion

The proposed neural network path planning algorithm provides the potential of massive parallelism in computation. This comes from the fact that the computation for the motion of each via point can be distributed to different networks, and thus can be executed in parallel. Furthermore, in our proposed algorithm, obstacles are not restricted to be convex and the penalty function can be automatically derived from the training process. This is because the penalty function associated with obstacles is generated by having the network learn the relationship between the x, y coordinates and its corresponding occupation state in the workspace, rather than by using a set of inequalities to approximate each obstacle. Although the training process for generating the penalty function of the workspace needs more than two hundred thousands sweeps of training on the simulation package, this situation could be improved by developing fast training algorithm or by using special hardware. As a result, the proposed neural network path planning algorithm seems to be suitable to the real-time planner in principle, in which the on-line replanning can possibly be done.

An arbitrary degree of precision of a path may be achieved by applying a sufficient number of via points. In other words, the number of via points should be chosen in such a way that the final path approximated by its via positions should not collide with obstacles while passing from one via position to its adjacent via position. With more via points, the distance between the adjacent via position gets smaller, thus could decreases the chance

of collision. In such a situation, the computation time would remain constant, even though the number of via points are increased. This is because the computation for each via point is totally localized, and thus can be performed in parallel.

The advantages that Park and Lee's method provides are:

1) the algorithm is potentially massively parallel;
2) arbitrary precision can be achieved by increasing the number of via points without increasing run time;
3) the use of the fast simulated annealing cooling schedule produces paths very rapidly.

Park and Lee's arguments about the use of simulated annealing to overcome the problem of local minima are questionable. We agree that the cooling schedule increases the performance of the algorithm, but we would argue that this is the result of introducing a slope to the flat areas in the energy surface.

Our conclusions about the method described are that it performs as well as Park and Lee's method, since the core algorithm is the same. Our main difference is that we have used the ability of a neural network to approximate any mapping to produce the distributed representation of the environment, rather than hand-crafting the network. In principle, a neural network is capable of this provided that there are sufficient units in the hidden layer. It was therefore felt that extensive testing of this ability was not required.

Chapter 4

Neural Network Navigator-1 — Design and Implementation

4.1 Introduction

In this chapter we focus on the generation of a steering scheme for the navigator. The task of this neural network navigator is to steer the robot towards each subgoal provided by the planner, while avoiding unexpected obstacles which are restricted to be stationary obstacles in this chapter. The steering commands are generated while the robot is moving, using feedback information about the locally visible environment. During this motion, the robot behaves independently from the planner until the planner has to be called in cases that the navigator is unable to handle.

With the assumptions we stated in section 1.2 of chapter one, the robot is considered to have a servo control system that is capable of following any reference trajectories. Therefore, in this chapter, we are only concerned with the problem of steering decision-making rather than the details of low-level control. When a subgoal is given, a steering command is generated at each sample interval by the navigator based on the local feedback information. The command is then applied to the reference model of robot's actuation system to produce a reference trajectory. Note that the transfer function of the reference model is assigned to 1, since the limitations imposed by the actuation system of the robot are not considered in this thesis.

In this chapter, we propose a neural network based steering scheme for the navigator. We assume that the environment changes unexpectedly, therefore

the sensor-based steering decisions must be made adaptively and in real-time. In conventional systems, sensing and action are treated independently in separated subsystems. The sensor-based guidance is often done by pre-establishing an explicit model about the relationship between the specific sensor output and the steering command variables. However, problems will arise when unforeseen changes happen to the geometrical parameters of the environment, or to the sensing or mechanical parameters. In addition, sensing often introduces undesirable feedback delays, as the result of sensor data-processing requirements (e.g. video image processing). In contrast, neural networks offer the possibility to replace a significant amount of modeling by adaptation and learning and thus, exhibit their flexibility in unexpected situations. More specifically, as we have described in section 2.2.1 of chapter two, the feedforward multilayer networks can approximate arbitrary nonlinear functions, so the sensor-based guidance can be achieved efficiently by neural networks which lend themselves as flexible "function approximators" between the sensing and action. In this way, the sensing and action can be integrated directly so that sensor-based steering decisions can be made in real-time in response to the dynamic world.

In the following sections, we first briefly review the neural network based approaches for sensory motor integration, and then develop a local steering scheme based on a neural network for the navigator in which steering commands are generated by directly converting sensor readings through the network. Finally, we present test examples which examine the behaviour of the navigator.

4.2 Neural Networks for Sensory Motor Integration

Sensor-based coordination of movements observed in biological organisms have recently been recognized as important for the control of artificial devices

(i.e. robots) (Sparks and Nelson, 1987). To achieve autonomous performance, it is important for robots to be capable of integrating sensory and motor information very flexibly in coordinated movements, as in biological systems. Neural networks seem to provide a most natural framework to exploit the sensory motor integration for the control of sensor-based movement.

Tolat and Widrow (1988) use an adaptive linear element (ADALINE) with visual inputs to balance an inverted broom-stick. In their system, the controller applies a positive or negative fixed impulse, depending on the broom state which is extracted directly from time sequences of crude visual images. During training, the input to the controller network consists of two 5 by 11 pixel images, one representing the present visual image of the inverted broom and the other representing one at a slightly earlier time. The desired output of the network controller is the corresponding impulse. Their simulations have shown that the neural network controller can successfully keep the inverted broom from falling only based on the observed visual images. In this system, the appropriate control signals–impulses are derived directly according to the observed visual images of the broom by means of the controller network.

Kuperstein (1988, 1991) reports on a neural network method for performing robotic sensory-motor coordination. His work is based on that of Grossberg and Kuperstein's (1986) adaptive sensory-motor control, in which topographic mappings found in biological motor control "modules" are proposed to use for robot sensor-based control. The system is designed to teach a three-joint robot arm to reach objects whose positions and orientations are located by two cameras. No kinematic relationships nor the calibration of joint angles to actuator signals are known *a priori*. In his system, the video images are fed to the input layer, whose outputs are connected to three arrays which convert the visual inputs into distributions in terms of camera orientations and their

disparity. These distributions are correlated with desired or target location in the target map with adjustable weights. Through training, the neural controller learns about a spatial representation by associating how the robot arm moves with what the cameras see. Kuperstein has demonstrated that the neural controller, called INFANT, can not only accurately reach the cylinder that it was trained on, but can also accurately reach other elongated objects that are arbitrarily positioned anywhere in space within the arm's reach.

Ritter, Martinez and Schulten (1989, 1990) present a different approach to the above visuomotor coordination problem dealt by Kuperstein. They applied the Kohonen's self-organizing feature mapping algorithm for the construction of topology conserving mapping between the camera"retinas" and neural lattice, as well as between the neural lattice and the joint space. An object is presented randomly to the system to induce camera retina coordinates and the corresponding neuron in the neural lattice. The joint angle vector associated with the neuron is applied to the robot arm, so as to compute the error between the camera retinal coordinates of the presented object and that of the robot end effector. Learning is based on the Widrow and Hoff (1960) type error correction rule. Note that since the neural networks are able to achieve adaptive hand-eye coordination, neural networks would have potential capabilities for sensor-based steering.

Pomerleau (1989) designed a very large modified 3-layer backpropagation network for the task of road following. The input layer is divided into three sets of units: two "retinas" and a single intensity feedback unit. The two retinas correspond to the two forms of sensory inputs, and consist of 30 by 32 and 8 by 32 units for receiving input from a video camera and a laser range finder respectively. Each of these 1217 input units is fully connected to the hidden layer of 29 units. The output layer consists of 46 units, and using 45 units to indicate 45 directions. The final output unit is a road intensity feedback unit

which is connected back to the intensity unit in the input layer to provide rudimentary information about the relative intensities of the road and nonroad in the previous image. One point to note is that Pomerleau's system aims at independent road following. Thus, a large amount of information on the overall detail of current road conditions have to be processed at each sampling time. Our aim, however, is to build a local steering strategy for the navigator that can be incorporated with the planner and generate reflexive actions by only using local feedback information at each sampling time.

Nagata, Sekiguchi and Asakawa (1990) built a structured hierarchical network control system for toy robots. A group of robots are used to play a cops-and-robbers game. The toy robot has four wheels and moves about freely. Twelve sensors are used to monitor internal conditions and environmental changes. Capture robots search for ultrasonic waves and infrared rays from escape robots and track them. Escape robots attempt to escape from the rays emitted by capture robots. The sensor signals represented as binary data are presented to the input layer of the structured hierarchical network, and output is used as motor control signals. The network model combines two backpropagation networks, a reason network and an instinct network. Through training, the robots learned behaviours such as capture and escape. In this system, the binary representation of the sensor readings simplifies the task of playing a game.

As described above, various problems have been solved based on sensory motor integration. In these neural network based systems, neural networks are used to relate sensing directly to action/control by learning from a set of examples or training patterns which are constructed in terms of what the sensors (i.e. cameras) see and the corresponding action. In this way, the processes about system parameter identification, sensor data-communicating and sophisticated algorithmic control signal calculation are no longer required.

Sensor-based control becomes computationally simpler since sensor readings can be directly converted into control commands. Furthermore, this computation is carried out in parallel due to the inherent parallel structure of neural network yielding high-speed advantages. Thus, in these systems reflexive pattern-driven responses can be implemented in real-time feasibly. For the sensor-based steering, steering commands should be generated by the navigator at each sampling time to drive the robot towards the current subgoal while avoiding collisions with obstacles. The navigator tends to achieve real-time reflexive pattern-driven responses at "servo rates" in the presence of dynamical changes in the navigation environment. Therefore, it is natural to extend the approaches of sensor-motor integration to solve the problem of sensor-based steering. In the next section, the proposed neural network based steering scheme for the navigator is presented.

4.3 Neural Network Navigator-1

In this section, we present the proposed local steering scheme for the navigator in which the steering decisions are made by directly converting local sensor readings through a backpropagation network. The local steering scheme can operate at "servo rates" in real-time. This is because at each step, the amount of information collected from the local environment and processing that is needed is small. Furthermore, the information is processed very efficiently due to the inherent parallel structure of the network. Note that the navigator described in this chapter is the first version which only deals with the simplified situation where no moving obstacles are in the navigation environment. The second version of the navigator, to be described in chapter six, can handle the more complex situation where unexpected moving obstacles exist in the navigation environment.

In the following subsections, we first introduce some notations and

assumptions used in the subsequent subsections. We then describe the neural network based steering scheme in detail. Finally, we show the architecture of the navigator and present the training process respectively.

4.3.1 The neural network based local steering scheme

As specified in section 1.2 of chapter one, the space to be navigated contains a finite number of unexpected obstacles which are modelled as convex polygons in the 2D plane. We assume that the minimum distance between points on any two obstacles is greater than the physical width of the robot, ϕ, plus a safety margin defined by the constant sf. Here, ϕ is defined as the diameter of the smallest circle encompassing the robot. This assures that a feasible path exists for the robot.

During navigation, the environment may change unpredictablly since uncharted objects can be put anywhere in the space. If there is an object on the planned path and in front of the robot, the navigator must modify the planned path by steering robot to follow the sides of the obstacle at a predefined distance (i.e. sf/2) until the originally planned path is again reached. The proposed local steering scheme is analogous to a blind person walking who takes one step at a time and then checks for the presence of obstacles in his path. In the local steering scheme, steering decisions are made based on a local grid. The local grid is used to collect the local information surrounding the robot through sensory feedback (i.e. video image). Based on the occupation state of obstacles in the local grid, a steering command is issued to direct the robot to avoid obstacles and move towards the subgoal indicated by the planner. Here, a neural network is used as a converter to directly transfer the sensory information from the local grid into appropriate steering commands at each sampling time. Since a local steering scheme should operate at "servo rates" in real-time, the representation of local grid and the avoidance behaviour should

be very simple and local to achieve the real-time performance. Next, we describe the construction of local grid in detail.

The local grid

During navigation, only information about the immediate environment surrounding the current position of the robot is needed. This local information is represented as 15 equal squares which are used to indicate the occupation state of the immediate neighbourhood of the robot. A binary value is assigned to each square; the value '1' implies that the square is fully or partially occupied, and '0' means that the square is unoccupied.

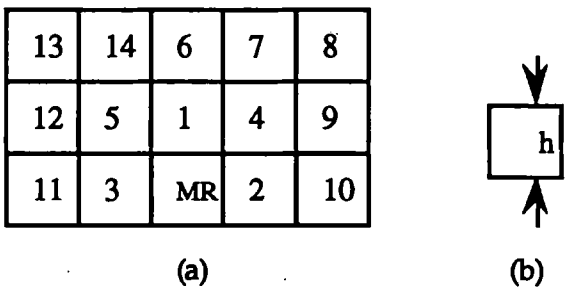


Figure 4-1. Local grid (a) 15 squares (b) Size of each square $h \geq (\phi+sf)/3$.

The local grid is illustrated in Figure 4-1(a). MR represents the current position of the mobile robot. The second level of squares (square 6–14) is used to detect the presence of obstacles by indicating the occupation status of the squares. The squares in the first level (square 1–5) nearest to the robot must be obstacle-free positions. This means that there must be at least three squares in the grid between any two obstacles. Since the minimum distance between two obstacles is defined to have a width of $(\phi+sf)$, the size of a square is therefore greater than or equal to $(\phi+sf)/3$, as can be seen in Figure 4-1(b).

Steering commands

There are six steering commands which define the following moving actions: forward (FW), backward (BW), turn left (TL), turn right (TR), stop (ST) and go subgoal (GS). With reference to Figure 4-1(a), to move the robot from MR to square 1 in the local grid requires a forward motion, thus the steering command FW should be issued. Similarly, to move to square 4 or 5 needs commands TR or TL respectively. This is because we have assumed that the robot can turn a maximum of 45° in one step. Therefore, to reach square 2 or 3, the robot needs to go back first and then turn to the right or left respectively. This corresponds to the combination of the commands BW and TR or BW and TR respectively. As we have stated before, a path consists of a sequence of subgoals. During navigation, the current subgoal is provided by the path planner and always exists. If there are no obstacles across the path, i.e. if all of the second level of squares are unoccupied, then the robot just goes towards the subgoal. It is in this situation, the command "go subgoal (GS)" is used.

From what we just described, it can be seen that each position in the 15-square local grid corresponds to a specific action for local piloting of the robot. The local steering scheme thus is based on the association between the presence of the obstacles indicated by the local grid and corresponding steering decision. By directly integrating the sensory information and steering commands, neural networks make implementation of the sensor-based steering scheme in real-time feasible and in a very straight forward way.

The network converter

Neural networks provide a natural tool to implement the sensor-based steering scheme. As specified in section 2.2.2 of chapter two, the multilayer feedforward type of neural networks (also referred as backpropagation nets) can

be trained to learn different relationships between variables regardless of their analytical dependency. Furthermore, this learning can be enhanced by the associativity and generalization properties of the neural networks. Therefore, the networks can be used as flexible "function approximators" to directly relate sensing and action.

For the problem of sensor-based steering, the backpropagation networks can be used as a converter by directly transferring the sensing input to the steering output vectors. During training, the network converts the input signals to output commands. It then compares the actual and the desired outputs and adjusts the connection weights to reduce the difference between them. Training is accomplished when the correct mapping between the input and the desired output has been achieved. In principle, the architecture of the network provides a direct converter between sensing and action. Therefore, the local steering scheme can be derived by having the navigator network learn the association between the state of the local grid and the corresponding steering commands.

4.3.2 Navigator network architecture

The architecture of the navigator network consists of a single hidden layer backpropagation network, as shown in Figure 4-2. The input layer consists of three units, and three groups of squares from the local grid are fed to the input layer by coarse coding (Hinton, McClelland and Rumelhart, 1986; Massone and Bizzi, 1989). As illustrated in Figure 4-3, each unit in the input layer has a 'receptive field' over the array of squares. The receptive field of the middle unit contains eight squares (Nos. 1-7 and 14 but not including MR) and is partially overlapped with the neighbouring fields of the other two units. The receptive field of the other units contains six squares and is partially overlapped with the receptive field of the middle unit. The activation of each

unit is the sum of the values of the squares belonging to its receptive field.

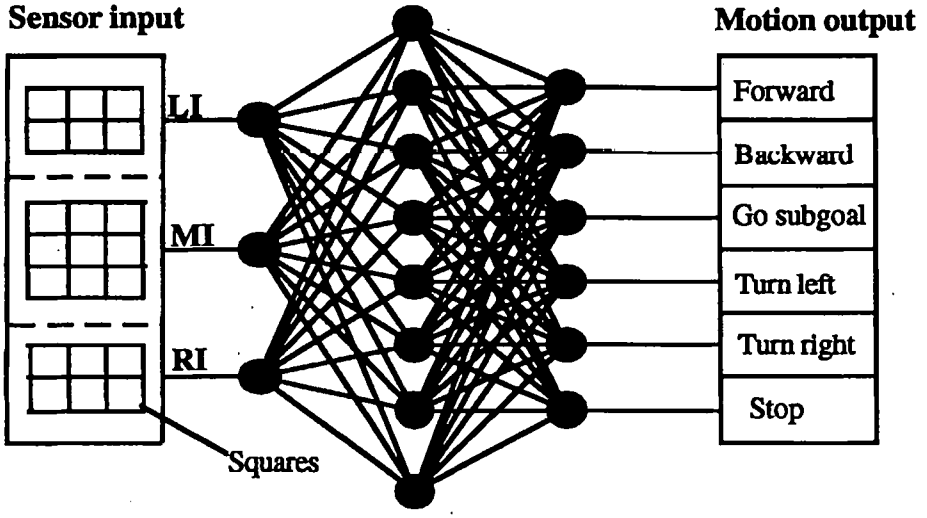


Figure 4-2. Architecture of the Navigator-1

The reason for using coarse coding is that the particular square occupancies in the local grid that cause different behaviours/actions are relatively sparse, since the environment is uncluttered and also very limited local information is used by the location grid (only using the squares in the second level to represent the presence of obstacles). The receptive fields of coarse coding give the advantage over a local encoding that it uses the information capacity of the units more efficiently by making each unit active more often.

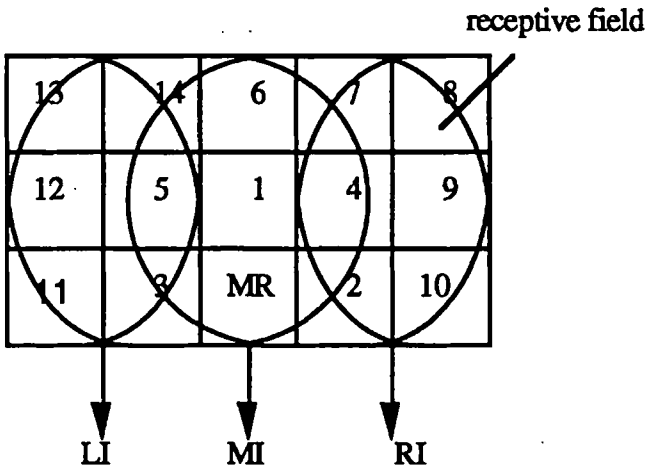


Figure 4-3. Coarse coding (LI is left input, MI middle input, and RI right input)

The output layer of the network consists of six units which correspond to the six steering commands: FW, BW, TL, TR, ST and GS. The command signals are represented as 1 bit each (0 or 1). The number of hidden units is empirically fixed to eight by using the simulation package as explained in chapter three.

4.3.3 Training

To train the neural network navigator, the network is presented with three groups of squares which are superimposed on the current video image. The corresponding steering command is supplied as the desired output. The set of training patterns are constructed by enumerating all the possible situations in terms of what kinds of visual patterns are likely in the local grid, and what the corresponding actions are. Figure 4-4 shows a example for generating a pair of training patterns.

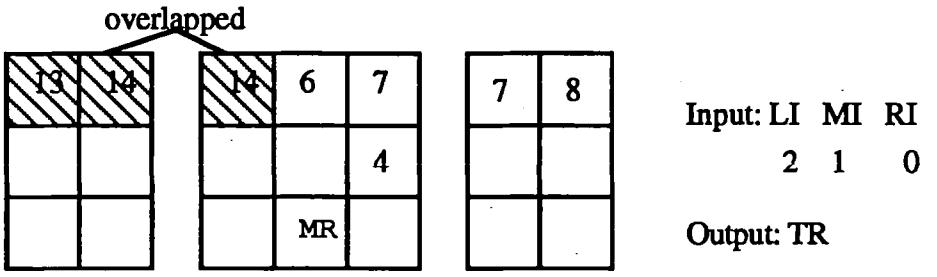


Figure 4-4. A example of a pair of training patterns.

In Figure 4-4, shaded squares represent the space occupied by the obstacles, the steering command TR will direct the mobile robot to the position 4. Note that the minimum distance between any two obstacles in the environment is assumed to be greater than a constant ϕ+sf, which equals to 3 times the size of the square. Thus, at most two objects can be present in the local grid at the same time. Furthermore, because the squares in the first level of the local grid (No. 1-5) are obstacles-free positions, the presence of obstacles only represented by the second level squares (No. 6-14) are therefore enumerable in the local grid. Table 4-1 lists 23 training patterns.

During training, the generalized delta rule with a momentum term is used to adjust the weights, and the sigmoid function is chosen as the transfer function for each of the units. Training is accomplished by using the simulation package.

TABLE 4-1. Training patterns

No	INPUT (square value)			OUTPUT					
	LI	MI	RI	BW	FW	GS	TL	TR	ST
1	1	3	2	1	0	1	0	0	0
2	2	3	1	1	0	1	0	0	0
3	1	3	1	1	0	1	0	0	0
4	2	3	2	1	0	1	0	0	0
5	0	1	0	1	0	1	0	0	0
6	1	1	1	1	0	1	0	0	0
7	1	1	2	1	0	1	0	0	0
8	2	1	1	1	0	1	0	0	0
9	2	2	1	1	0	1	0	0	0
10	1	2	2	1	0	1	0	0	0
11	1	2	1	1	0	1	0	0	0
12	0	1	1	1	0	0	1	0	0
13	0	2	2	1	0	0	1	0	0
14	0	2	1	1	0	0	1	0	0
15	1	1	0	1	0	0	0	1	0
16	2	2	0	1	0	0	0	1	0
17	1	2	0	1	0	0	0	1	0
18	1	0	1	0	1	0	0	0	0
19	0	0	0	0	0	1	0	0	0
20	0	0	1	0	0	1	1	0	0
21	0	1	2	0	0	1	1	0	0
22	1	0	0	0	0	1	0	1	0
23	2	1	0	0	0	1	0	1	0

As described in chapter 3, one of the important steps in training a network is to set the appropriate learning rates and momentum terms for two stage of training: setting relatively larger learning rate and momentum at the beginning to get fast convergence, and setting smaller learning rate at later to achieve accurate approximation that converges to desired learning patterns with small errors. It is also known that using different learning rates for each layer in a multilayer network can decrease the learning time. In particular, having a larger learning rate for the hidden layer rather than for the output

layer allows the hidden layer to form feature detectors during the early stages of training. These feature detectors can then be combined to form more complex detectors at the output layer.

At the start of the training, we set the momentum coefficient to 0.8, the learning rate of the hidden layer to 0.6 and 0.45 for the output layer. After 1200 sweeps, the momentum coefficient is reduced to 0.4, and the learning rate of the hidden layer is decreased to 0.2 and 0.1 for the output layer. After 2000 sweeps of training on the 23 training patterns, the network satisfactorily converges to the desired patterns with the RMS error 0.0156. The learning curve and the weights distribution of training are shown in Figure 4-5.

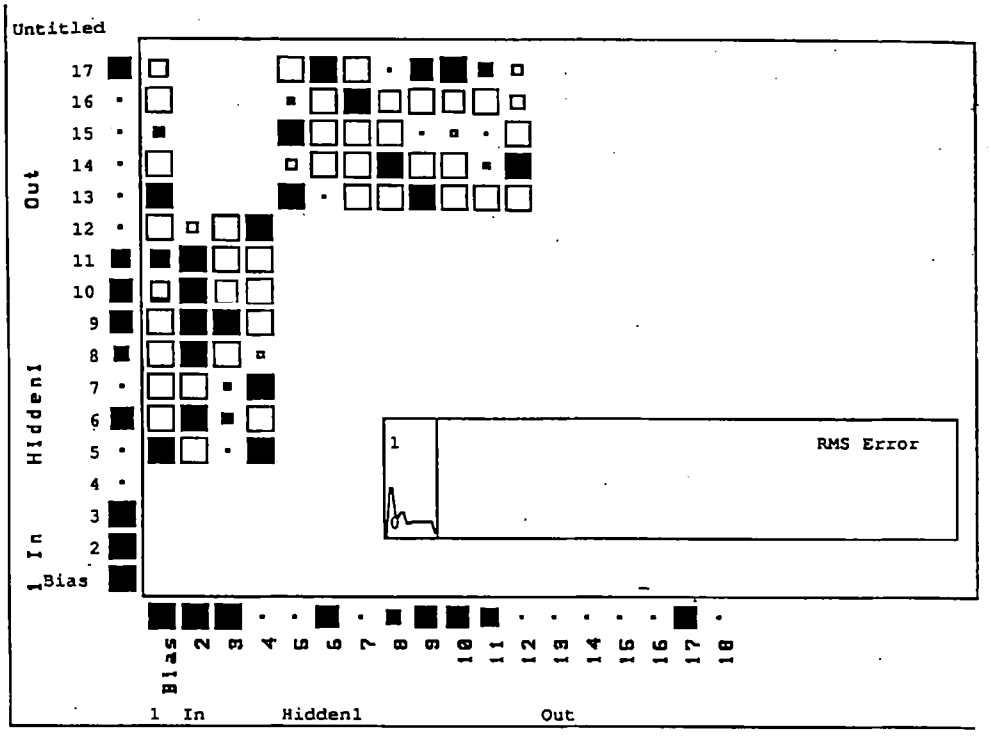


Figure 4-5. Training curve and weight distribution

The network which associates the local sensory information with the corresponding steering command is thus derived with learned connection weights as shown in Table 4-2 (a), (b). When an input pattern collected from

the local grid is received, the trained navigator network generates an output vector in which each continuous value equals 1 or 0 approximately and corresponds to a specific steering command.

TABLE 4-2(a). Hidden layer connection weights and bias terms

Input Layer	Hidden Layer							
	Node1	Node2	Node3	Node4	Node5	Node6	Node7	Node8
Bias	2.7039	-2.7990	-3.5164	-1.1560	-2.0201	-0.8426	0.7861	-1.1977
Node1	-2.6118	2.2719	-6.8129	1.9665	1.5380	1.1725	1.3157	-0.4302
Node2	0.0422	0.4697	0.2887	-6.0551	6.2279	-0.9736	-4.4207	-6.0320
Node3	5.9622	-5.1718	3.1027	-0.3896	-1.5604	-2.8797	-1.2822	2.0199

TABLE 4-2(b). Output layer connection weights and bais terms

Hidden Layer	Output Layer					
	Node1	Node2	Node3	Node4	Node5	Node6
Bias	1.0393	-1.9586	0.5514	-1.9076	-0.7151	-1.5975
Node1	1.0009	-0.5378	2.7052	0.2977	-4.2924	-0.4608
Node2	-0.2231	-1.4206	-3.1649	-1.5337	2.7731	-1.1107
Node3	-2.5646	-2.0811	-5.9092	6.3088	-1.6998	-1.0948
Node4	-3.3763	2.8455	-2.3982	-0.9842	0.2147	-0.8567
Node5	3.6395	-2.2313	-0.0295	-1.8432	0.9315	-0.2812
Node6	-1.8221	-1.3917	-0.4138	-0.9290	1.3546	-0.8553
Node7	-3.3333	0.3466	-0.2570	-1.0146	0.7053	-1.7268
Node8	-2.8944	1.7896	-2.5229	-0.6708	-0.5119	-1.0470

4.4 Test Examples

To test the navigator's behaviour, we simulated various navigations in a 2-D workspace which consists of 30 by 30 cell units with several obstacles popping up during navigation. The simulation is based on the trained neural network navigator, and trajectories of the robot are displayed on the screen of a PC-386. In our simulations, we intend to verify the capability of the navigator by demonstrating that the robot can reach any subgoal indicated by the path planner and respond adaptively to changing environmental conditions. This is done by first randomly chosing an initial position and a collision-free

subgoal within the simulated workspace. At each step, the local grid is used to collect local information for the navigator, the position of robot is then updated based on the steering commands issued by the navigator. A trajectory of the navigation is obtained by recording the history positions of the robot from the initial position to the subgoal. Each successful navigation is classified by examining its trajectory in terms of if it is collision-free and if its subgoal is reached. Next, we give two examples of the navigations. The simulation programs are written in Turbo C.

In the following simulation examples, the obstacle regions are shown by the solid filled areas; the trajectories of the robot are shown by shaded squares. The initial position and the subgoal are also indicated before a navigation starts. The robot is moving at a constant speed which is one unit at each step. It is also assumed that the sensors on the robot are capable of identifying each obstacle from the background and have no blind spots (i.e. seeing everything surrounding the robot). As the path planning has already been done according to the map of the workspace before the navigation starts, there are no obstacles on the planned path. The obstacles on the planned path shown in the examples are actually the uncharted obstacles which just pop up during navigation.

Figure 4-6 (a), (b) shows two trajectories of the robot with different subgoal positions but the same start position. Note that the environmental conditions are changed at each simulation, because different subgoals direct the robot to a different region of the workspace in which the geometrical parameters of the region presented are completely different. These two trajectories are typical of the many simulations that have been carried out on this workspace. Approximately 100 trajectories with different starting points and subgoals have been simulated with, as yet, no failures. We are therefore confident that this method is reasonably robust under these constrained circumstances.

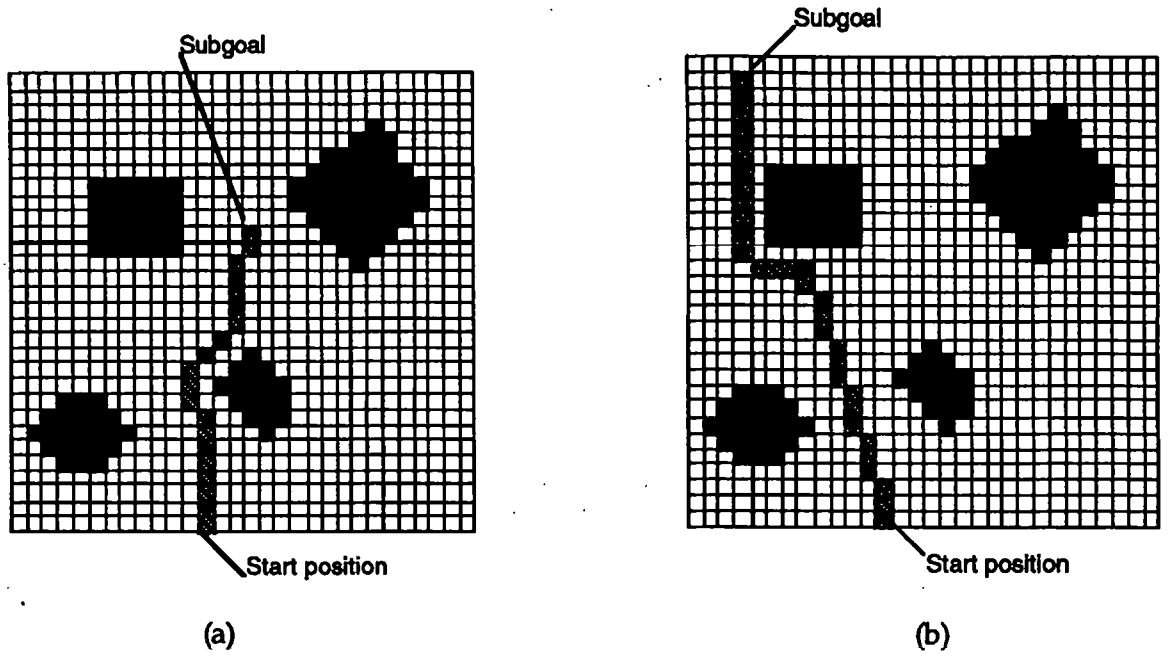


Figure 4-6. Trajectories of navigation (a) Trajectory 1 (b) Trajectory 2.

4.5 Summary

In this chapter, we reviewed work on neural networks for sensory motor integration, and described the first version of the navigator which steers the robot towards the current subgoal while avoiding unexpected collisions. Neural networks offer a distinctly different approach to sensor-based guidance by directly integrating the sensory and action information. The navigator is built by using a single hidden layer backpropagation network which is trained to learn the mapping between the sensory input and steering command output. In the navigator network, each steering decision is made by directly transferring the local sensory information 'a forward sweep' through the network. The simulation results seem to show that the navigator is capable of effectively guiding the robot to avoid collisions with unexpected obstacles. The navigator is also potentially capable of high-speed processing since it handles input patterns on a real-time basis. The real-time basis comes from two factors. At first, the input sensory information is taken from the local grid and thus

only consists of a small amount of information. Secondly, the inherent parallel structure of the network allows very efficient processing since determining the steering command from the sensory input merely involves a forward sweep through the network.

This version of the navigator, however, has some limitations. The network navigator is trained with all the possible training patterns which in effect is not necessary. The network navigator should determine its action in any environments by inferring from the patterns it has learned. This is because neural networks have an inherent capability of generalization. In an enhanced version of navigator to be described in chapter 6, we adopt an more efficient training method, where only necessary patterns are taught. This takes better advantage of the properties of neural networks.

In this version of the navigator, the steering command "Stop" is not actually used. This command is designed for the emergency cases which, however, are not dealt with by this version of navigator. This navigator is only a simplified version in which the moving obstacles are not considered. With moving obstacles, the navigation situations would be much more complex. For instance, when robot's sensors suddenly perceive a moving object crossing its path, meanwhile unexpected stationary obstacles block its way to move away from the moving obstacle, the command "Stop" will be used in this emergency situation.

If moving obstacles are considered, however, the steering decisions are made not only depending on the current observed position of the obstacle but also relying on the speed and direction of the obstacle in the near future. Therefore, motion prediction is desired here. In next chapter, we will introduce the *motion predictor* which provides a prediction about the movement of the obstacle at each sampling time.

Chapter 5

A Neural Network Motion Predictor

5.1 Introduction

In this chapter, we focus on the motion prediction of moving objects based on their observed trajectories in the past. The purpose of developing the predictor is to let a robot be endowed with a mechanism of detecting moving objects and estimating their motions in the near future. With this capacity the robot can make steering decisions appropriately to avoid collisions with moving objects whose motions are uncertain.

Predicting the motions of moving objects in an unconstrained environment is often a difficult task since moving objects can change their movements in any way and at any time. Before navigation starts, there is no information about moving objects. During navigation, objects whose motions are completely unknown to the observer, move into the robot workspace. This may disrupt the motion of the robot. The information about their movements can only be acquired through processing measurements obtained by sensing devices. Thus, motion prediction cannot be accomplished by using a pre-established/known analytical model. It has to resort to adaptive methods, which in effect construct some approximate model of the unknown process from a sufficiently long sample of sequence data.

One of the most widely used adaptive methods for predication is adaptive linear regression (Rhodes, 1971). The predicted value $\hat{x}(t+1)$ at the next time step is given in terms of a linear combination of a fixed number m+1 of

previous values, i.e.

$$\hat{x}(t+1) = \sum_{i=0}^{m} a_i \, x(t-i) \tag{5.1}$$

The initially unknown coefficients a_i, $i = 0 \ldots m$ are recursively estimated by a suitable stochastic minimization procedure for the mean square error of the predicted values (Rhodes, 1971). If the actual process can be satisfactorily described by a linear model, the coefficients a_i will gradually converge to a set of values providing a good approximation of the process. In the case of unconstrained motion, however, the underlying behaviour of moving objects varies with time, therefore, the linearity condition may not always be well obeyed. This prediction method is thus not suitable to general situations of motion prediction where a underlying motion could be a combination of linear and nonlinear processes.

On the other hand, neural networks render them prime candidates for nonlinear modeling and time series filtering. Their usefulness in this area stems from their inherent nonlinearity and capabilities of learning and generalization. It is thus possible to use neural networks to develop adaptive methods for motion prediction in situations where combinations of nonlinear and linear processes are presented. The predictor we developed adopts a neural network approach.

In the following sections, we first survey relevant work on using neural network approaches in the area of time series prediction. We then present the proposed on-line predictor which is based on the Elman net and an on-line learning and prediction scheme. Finally, the adaptive capabilities of the predictor are discussed through simulations of three typical examples.

5.2 Neural Networks for Time Series Prediction

It is known that neural networks can be used as arbitrary functional approximators (both nonlinear and linear) of an input space. Thus, neural networks seem to be particularly effective for time series prediction problem where the prediction is performed entirely by inference of future behaviour from examples of past behaviour regardless of the analytical dependency. We outline below some work based on this approach.

Lapedes and Farber (1987) use backpropagation networks with sigmoid activation functions for time series predictions. To demonstrate the use of the nonlinear neural net formalism, they choose to predict a complicated time series which is generated by iterating the classic logistic map (Feigenbaum, 1978). The network consists of one input unit, five hidden units and one output unit. In the output unit, a linear activation function is used. They train the network on 1000 sets of (x(t), x(t+1)) pairs and then use the trained network to predict one time step into the future for 500 additional points. The final prediction error is reasonably small. However, the prediction is based on a network which is trained by off-line training using 1000 training patterns.

Subsequently, Moody and Darken (1988) show that neural networks with radial basis function (RBF) can predict nonlinear time series much more accurately compared to Lapedes and Farber's results. However, it requires about ten times more training data to achieve comparable prediction accuracy. Moody and Darken (1988) point out that if data is cheap and plentiful, the RBF network would be preferred.

Walter, Ritter and Schulten (1990) suggest that Kohonen's self-organizing maps can be used to predict highly nonlinear time sequence data. The main idea is to use a Kohonen net to adaptively discretize the set of the input data,

and to estimate in each discretization cell a separate set of linear prediction coefficients. In their simulation, this method is applied to predict the trajectories of an object in a set of nonlinear potentials. The trajectories are chosen within the cube $-1 < x, y, z < 1$. The network is trained using a series of trajectories with randomly chosen starting values. The trained network can then make quite accurate predictions. However, a sufficient number of samples have to be exposed to the network before predicting and the trajectories of the objects must remain in the cube.

Lambert and Hecht-Nielsen (1991) explore the use of both recurrent and feedforward networks for the prediction of the plant state of a chemical process. The task of this work is to build predictive models for plant variables and compare the performance of the feedforward and recurrent networks on this prediction problem. Their experiment results demonstrate that the recurrent backpropagation network (Williams and Zipser, 1990) performed more accurately and could employ a longer interval than the time-delay feedforward networks (Waibel, 1989). However, training of the recurrent network with 8 hidden units requires about 2 days of computation on a Sun 4 workstation.

The methods for time series prediction described so far seem to provide acceptable performance for a range of problems. These methods, however, demand either a large quantity of training data or considerable amount of computation for the training process. In the situation of predicting the motion of moving objects, it would be impossible to make a prediction if we have to either wait a period until sufficient training data on the motion of the moving objects has been obtained or use large amount of time for the training process. This is because the motions of the moving objects vary with time. Prediction has to be carried out at each sampling time by using the most recent data on the motions of objects. Moreover, the result of prediction at each time will be

immediately used by the navigator to take appropriate actions to avoid the moving object. Motion prediction must thus be done on-line and in real-time. Consequently, the methods we described above are not suitable for motion prediction in this context.

In the next section, we present the proposed on-line motion predictor which is based on a recurrent network.

5.3 The Recurrent Network Motion Predictor

In our proposed method, motion prediction is done at each sampling time synchronizing with the navigator since the motion of an object may vary with time. Once a moving object is detected within a predefined distance from the robot, the prediction process starts. The process consists of two stages: at the beginning, the predictor network is trained briefly using a fast learning algorithm to get a reasonable amount of knowledge about the current motion of the moving object. The training patterns consist of only four most recent trajectories of the moving object, hence the time for each training cycle would be very short. The trained predictor network is then used to make a prediction of the position one time step ahead. In the mean time, on-line training is carried out to improve the predictor's performance and to achieve adaptive predictions.

To make an implementation of the proposed on-line prediction feasible, an Elman recurrent net is employed as the predictor network. Among the available recurrent networks, the Elman net (Elman, 1990) has the simplest architecture. It can also work with the standard backpropagation learning algorithm. Since its architecture and learning algorithm are computationally simple, the Elman net yields real-time basis for the on-line prediction. Furthermore, like any other recurrent networks, the Elman net possesses the

characteristic of a dynamic memory. This is an advantage for solving prediction problems. In common with any ordinary time series prediction, the motion prediction also requires knowledge of the preceding values in the underlying motion of the object. A recurrent network, in contrast to feedforward networks, is one in which feedback is allowed from the output of a neuron to the inputs of neurons within the same and other layers. Recursion in recurrent networks provides a means of propagation in which proceding activities can be sustained. In this way, the information about time is automatically stored in the networks as a dynamic memory. Recurrent networks are therefore suitable for motion predicting. In the Elman nets, a set of "context" units memorise some past states of the hidden units, so the outputs of the nets depend on an aggregate of the previous states and the current input. With the internal memorising mechanism, prediction can be done using merely current information about the motion of the moving object rather than using both some proceding and current information to represent the dynamic characteristic of the motion time series.

In the following subsections, we first describe the architecture of the predictor network. We then introduce a fast training algorithm and an on-line learning and prediction scheme which are employed to achieve on-line prediction, .

5.3.1 Architecture of the Elman net – Predictor network

A block diagram of the Elman net is shown in Figure 5-1. It can be seen that the main structure of the Elman net is feedforward. In addition to the input units, hidden units and outputs units, there are also context units. External signals are received by the input units which are buffer units that only pass the signals without changing them. The output units are linear units which sum the signals fed to them. The hidden units have nonlinear activation functions. The context units are used to copy previous activations of the hidden units so

that activations associated with an event at time t-1 can influence processing at time t. Thus, the context units can be considered to function as one-step time delays which enable Elman nets to possess the characteristic of a dynamic memory.

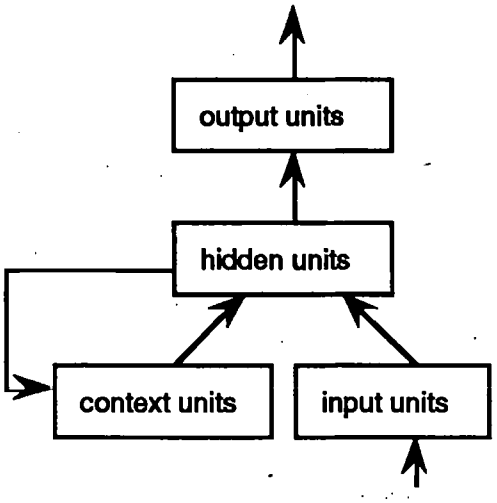


Figure 5-1. A block diagram of an Elman net

As shown in Figure 5-1, forward arrows represent sets of modifiable connections trained using backpropagation of errors. The backward arrow represents recurrent connections whose strengths are usually fixed to 1. Because the recurrent connections are not trainable, the Elman net is only partially recurrent.

The training process in an Elman net can be described as follows. At the beginning of training, the weights of the modifiable connections are set to small random values as in a feedforward network. For a sigmoidal activation function, the initial outputs of the context units are usually set to 0.5 which equals one half of the maximum range of the activation value that the hidden units can take. At a specific time k, the current input signals (at time k) and the outputs of the context units which are the previous activations of the hidden units (at time k-1) are used as inputs to the network. At this stage the network is a feedforward network in which the inputs are propagated forward to the

output layer to produce the outputs. Normally, the weights of the feedforward connections are modified using the standard backpropagation learning rule (Rumelhart and McClelland, 1986). At the same time, the outputs (at time k) of the hidden units are sent back through the recurrent links to the context units, and stored there for the next training step (time k+1).

The detailed architecture of our predictor is shown in Figure 5-2. It consists of two input units, five context units, five hidden units and two output units. The inputs to the network are the velocity value and the moving direction at the current sample time. The network outputs the predicted values of the velocity and the direction one time step into the future. The number of hidden units is empirically fixed to five. The number of context units is the same as for the hidden units, since the context units are used to copy the previous states of the hidden units. In the figure, the feedback connections indicated by a dashed line are not trainable, and fixed to 1.

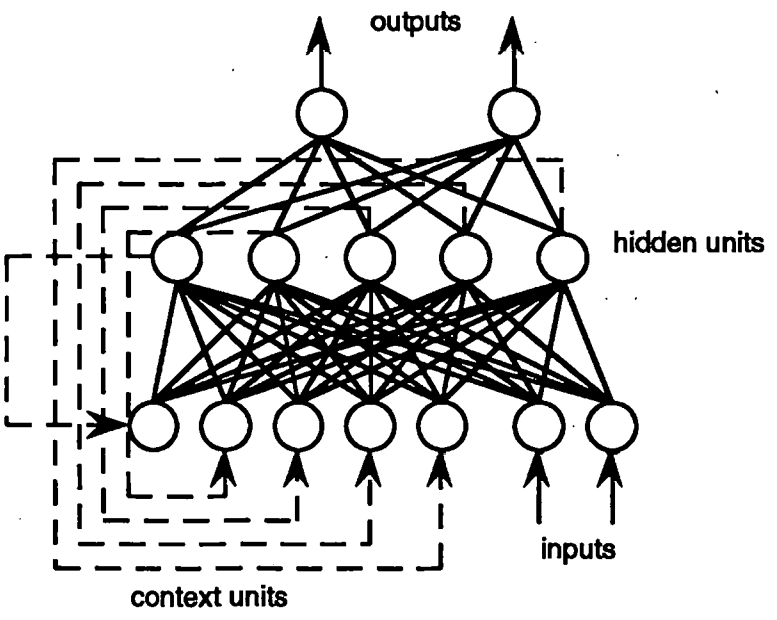


Figure 5-2. Architecture of the Elman net predictor

In order to meet the requirements of on-line training, we use a modified backpropagation learning algorithm to speed up the training process. We will

describe this fast learning algorithm in the next subsection.

5.3.2 A fast learning algorithm – Pseudo-impedance control

To develop a learning algorithm with fast convergence, we employed pseudo-impedance control as described in Nagata, Sekiguchi and Asakawa (1990). Generally, supervisory learning in a neural network is accomplished by adjusting the connection weights to obtain the desired pattern in response to a given input pattern. The standard backpropagation learning algorithm changes the connection weights based on a driving stimulus, which is the gradient of the error metric passed through a first-order filter. As we have described in subsection 2.2.2 of chapter 2, this is done by first calculating the difference between the actual and desired outputs to obtain the total error term at the output layer. The gradient information is then calculated at each layer so that the global error term E(t) can be minimized over the weight spaces of the network. The result of the learning should converge to the global minima as described in Hecht-Nielsen (1989). However, local minima may slow down the learning processing or even stop learning at a local minimum.

To decrease the possibility of trapping in a local minimum and achieve fast convergence, the pseudo-impedance control is employed as the learning law. It is derived through an analogy to a mechanical vibration system. The idea is that the behaviour of a network in the learning process can be treated as a second-order damped vibration system that has forced power $(-\partial E(t)/\partial w(t))$ (Nagata, Sekiguchi and Asakawa, 1990). By providing the network with damped vibration characteristics, the number of training lessons (weight updates) can be reduced:

$$Jd^3w(t)/dt^3 + Md^2w(t)/dt^2 + Ddw(t)/dt = -\partial E/\partial w(t) \qquad (5.2)$$

$\Delta w(t)$ can be derived by approximating discretely from the quadratic differential equation (5.2),

$$\Delta w(t) = \{-\Delta t^2 \partial E/\partial w(t) + (2J+M\Delta t)\Delta w(t-\Delta t) - J\Delta w(t-2\Delta t)\} / (J+M\Delta t+D\Delta t^2) \quad (5.3)$$

where $\Delta t = 1$, then

$$\Delta w(t) = -\eta \partial E/\partial w(t) + \alpha\Delta w(t-1) + \beta\Delta w(t-2) \quad (5.4)$$

$$\eta = 1/(J + M + D)$$
$$\alpha = (2J + M)/(J + M + D) \quad (5.5)$$
$$\beta = -J/(J + M + D)$$

where J is the jerk, M the mass, D the viscous damping coefficient, w the connection weight, E the sum of the squares of the total errors, Δw the weight change, t the number of training lessons (weight updates), and η, α, and β are the learning parameters.

Pseudo-impedance control can be implemented by storing the weight changes from the two most recent lessons. The connection weights are adjusted using a second-order gradient filter. When β is 0, pseudo-impedance control is the same as error backpropagation. It is not difficult to assign suitable values to the learning parameters since the relationship among the learning parameters η, α, and β is expressed using parameters J, M, and D. The necessary conditions of η, α, and β can be derived from the constraints $J \geq 0$, $M \geq 0$, and $D \geq 0$, which ensure the stability of the second-order gradient filter.

In our experiments, the pseudo-impedance algorithm provided a method which converged to a solution, on average, 10 times faster than standard

backpropogation. This increase in speed means that there is a greater chance of converging to a solution in a limited number of steps (or a limited amount of time). The proposed predictor must have this ability.

5.3.3 On-line prediction and learning

As we stated at the beginning of this section, motion prediction needs to be done on-line and in real-time. Furthermore, it should be improved by carrying on learning to achieve adaptive performance while the behaviour of a moving object varies with time.

An on-line learning and prediction scheme is utilized by the predictor network to adapt its predictions with time. Learning (training) of neural networks can be either on-line or off-line. Off-line learning attempts to calculate weights without any references to time ordering of the training data. Thus, all the training data must be collected before learning starts. On-line learning, on the other hand, attempts to modify weights as information in the form of training data flows in. On-line learning is thus able to modify the behaviour of the predictor network adaptively in the presence of unexpected changing conditions.

To achieve on-line learning and prediction, we adopt the idea of specialised learning. Specialised learning was originally proposed for the on-line identification and control of physical dynamic processes (plants) (Psaltis, Sideris and Yamamura, 1988; Saerens and Soquet, 1991). In specialised learning, the neural controller learns no longer from input-output pairs but from a direct evaluation of the network accuracy with respect to the output of the plant. There is no longer a specific training stage in the specialised learning. Moreover, the network learns continually and is therefore adaptive. In our on-line learning and prediction scheme, we make a prediction one time

step ahead $V_p(t+1)$ by using the current state of moving object V(t), which is obtained from the underlying actual motion series. At the next sample time, by observing what actually occur V(t+1), we calculate the difference between the actual values and the predicted values, and then modify the weights of the predictor network by using the fast learning rule described in subsection 5.3.2. At the same time, we use the actual observed value V(t+1) to make the next prediction. Such an iterative procedure is used for the on-line learning and prediction.

Note that the actual "past" data is required to perform the prediction, i.e. V(t) is obtained from the actual motion time series rather than from the result of the last prediction. The capital variable V represents the speed vector which consists of the value of speed and the angle of the moving direction.

The whole procedure of the on-line learning and prediction can be summarized as follows:

(a) The predictor network receives the first four sampled pairs of motion vectors. The network is trained briefly by using the pseudo-impedence control learning algorithm to get *a priori* knowledge about the current motion of an object. The training often takes about 200-300 iterations, thus it only requires a short run time.

(b) At time k, when the current state of the moving object V(k) is received, the predictor network outputs the predicted state $V_p(k+1)$. Note that the time required for a prediction is quite short, since making a prediction only involves a forward sweep through the simple predictor network.

(c) At time k+1, when the actual observed values V(k+1) of the motion is received, the following processes are taken: calculating the prediction error:

$V(k+1)-V_p(k+1)$; calculating the weight changes: $\Delta w(k+1)$ through the network by using the pseudo-impedance control learning algorithm; updating the weights of the predictor network and making next prediction: $V_p(k+2)$;

(d) At time k+2, repeat the processes in the (c).

The procedure is repeated until the moving object is impossible to disrupt the motion of the robot (i.e. the distance between the moving object and the robot is up to a predefined constant which will be described in later chapters).

5.4 Test Examples

In this section we present the results of three simulations, which are done by applying the method described above to particular prediction tasks. To test the effectiveness of the predictor in the presence of a range of nonlinearities, we choose three typical motion series:

a circular motion with a constant speed;
a second-order dynamic motion of a object;
and a chaotic time-series.

For the first two examples, we predict the motions of an object in a 2D environment, since the predictor is designed to be used in 2D navigation environments. The architecture of the predictor for these two cases is the same as that shown in Figure 5-2. In the third example, the motion is only relevant to the y-axis direction, in which the architecture of the predictor needs minor modification. In each of the simulations, we randomly choose an initial position for a moving object, the rest of the trajectory of the moving object is then generated by a predefined equation. The first four positions of the moving object are used to train the network predictor briefly, using the

pseudo-impedance control learning rule. The prediction is then carried out by using the on-line learning and prediction scheme. The simulation is implemented in C programming language.

Note that in the simulated predictions, the size and shape of the moving object are not involved by assuming that the size of the moving object is small enough compared to the robot and the environment. The moving object thus can be treated as a point.

a) Predicting a circular motion

The purpose of this simulation is to predict a circular motion. The circular trajectory is given by the equation:

$$[x(t) - 6]^2 + [y(t) - 6]^2 = 16 \qquad (5.6)$$

The inputs to the predictor network, as shown in Figure 5-2, are the velocity value of the line segment velocity and its angle of the moving direction (-180° to +180°). They are calculated from the most recent two observed positions of a moving object. The outputs of the predictor are the predicted values of the velocity and the direction angle at next sample time.

In this example, a moving object starts a circular motion with a constant speed ($\pi/3$ units) from a randomly chosen initial position: (7.92, 2.49) in the circular trajectory. When the fourth position of the moving object is obtained, the predictor then starts training to acquire certain knowledge about the current motion of the object. By using the pseudo-impedance control learning algorithm, the training takes 300 iterations with only the four past positions of the moving object. The trained predictor is then used to predict the circular motion one time step into the future. At the forthcoming sampling time,

when the new actual position of the moving object is observed, the last prediction error is then calculated and back-propagated through the predictor network once. Meanwhile, the new prediction is done based on the network with new updated weights. Figure 5-3 illustrates the results of 16 predictions.

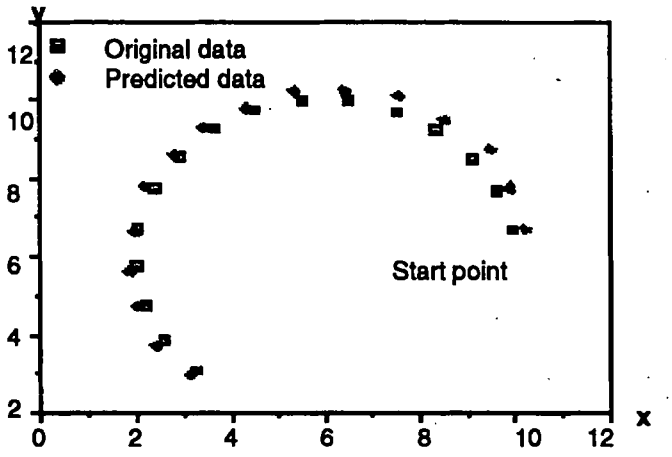


Figure 5-3. Prediction of a circular motion

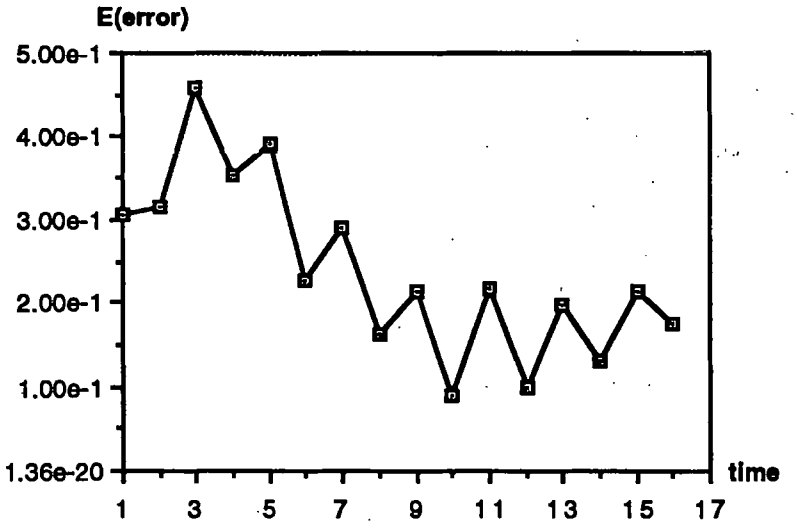


Figure 5-4. Prediction errors

In this simulation, the learning parameters are set to η (the learning rate) = 0.1, α (the momentum term) = 0.4, and β (see equation (5.4)) = –0.2. To show the improved predictive accuracy with increasing time t, we define the prediction error E as the distance between a predicted position of a moving object and its

actual position:

$$E = \sqrt{(x_{t+1} - \hat{x}_{t+1})^2 + (y_{t+1} - \hat{y}_{t+1})^2} \tag{5.7}$$

The results of prediction errors for 16 positions are shown in Figure 5-4.

b) Predicting a second-order dynamic motion

The task of this simulation is to predict the motion of a chair on a flat 2D floor when it is pushed by a force F_0 at the beginning. The initial position of the chair is randomly chosen at (0,0), the initial force at x and y directions ($F_0(x)$, $F_0(y)$) is set to 5 units respectively, the mass of the chair m is set to 1 unit and the sliding-floor friction coefficient μ is set to 0.05. The motion of the chair can be described by a second-order dynamic system:

$$\begin{aligned} F_0(x) - \mu^* m^* g &= m^* \frac{d^2x}{dt^2} \\ F_0(y) - \mu^* m^* g &= m^* \frac{d^2y}{dt^2} \end{aligned} \tag{5.8}$$

When the initial force is applied to it, the chair is given an acceleration and starts moving. During the chair's movement, the friction force between the chair and floor is imposed on the chair. Thus, the speed of the chair gradually decreases and eventually it would stop moving. During this motion, the chair moves straight along the direction of the initial force. The architecture of the predictor used in this simulation is the same as the one used in the first simulation, so are the learning parameters and the procedures of the prediction. The sampling time interval is one time unit. The results of 7 predictions are shown in Figure 5-5.

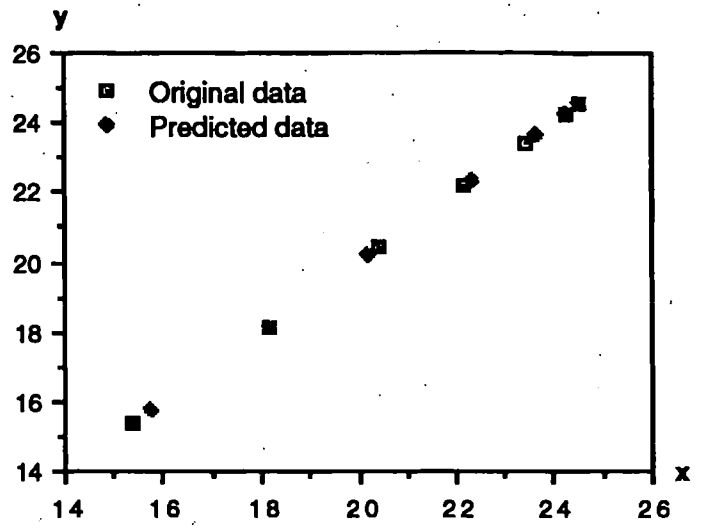


Figure 5-5. Prediction of the movement of puching-chair

It can be seen from the figure that the predictor quickly homes in on the motion of the chair and tracks it quite well. The error of the predictions reducing with time is illustrated in Figure 5-6.

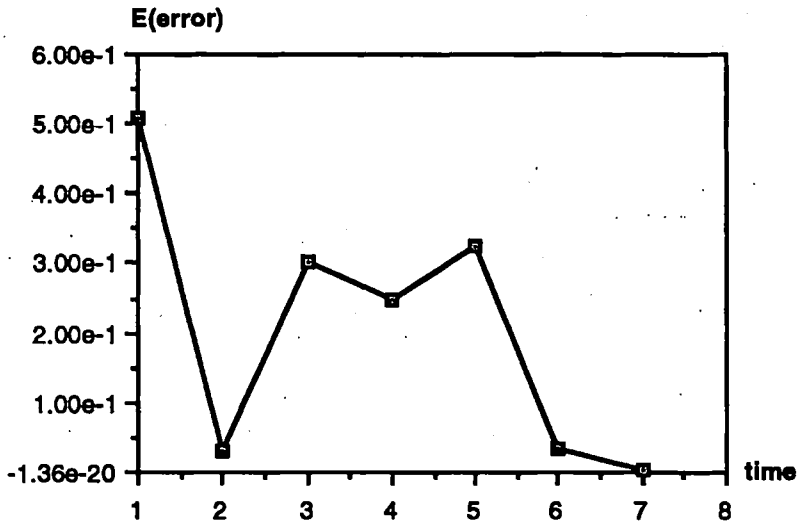


Figure 5-6. Prediction errors

c) Predicting a chaotic time series

To further illustrate the use of the nonlinear neural net formalism and test the predictive capability of our neural network predictor, we conduct the third

simulation. In this prediction, the time series is generated by iterating the logistic map described by Feigenbaum (1978).

$$y(t+1) = b * y(t) * (1 - y(t)) \tag{5.9}$$

where b is set to 3.97. This iterated map produces an ergodic, chaotic time series if b is chosen equal to 3.97 (Feigenbaum, 1978). The input to the predictor network is y(t), and the output is y(t+1). Since this series is only relevant to the y-axis direction, the architecture of the predictor network is modified as that shown in Figure 5-7.

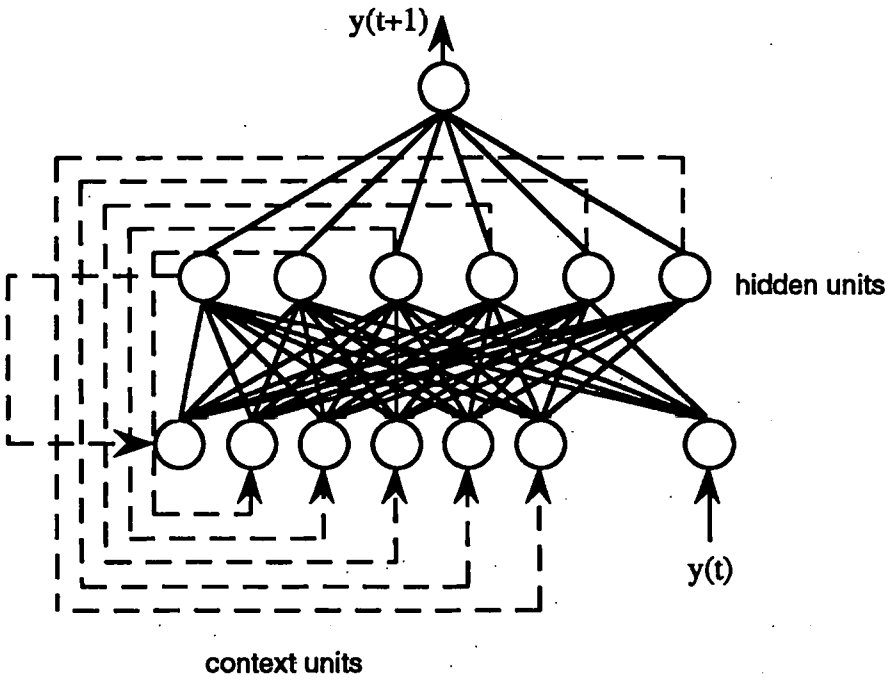


Figure 5-7. The modified architecture of the predictor for chaotic case

Instead of having two input units, two outputs units and five hidden and context units as shown in Figure 5-2, this network consists of only one input unit, one output unit, and six hidden units and context units.

The initial value of y is randomly set to 0.4. Similarly to the procedure of the first simulation, the predictor network is trained using the fast learning rule

on 4 sets of (y(t), y(t+1)) pairs. The trained network is then used to predict one time step into the future for additional 45 points. Meanwhile it refines its prediction by carrying on a learning cycle for each trial. The performance of the prediction is shown in Figure 5-8. It can be seen that the network predictor manages to adaptively follow the underlying attitude of the chaotic time series. The error of predictions is given in Figure 5-9.

5.5 Summary and Discussion

In this chapter, we have described the motion predictor, which is based on the Elman recurrent network. In the context of robot navigation, the motion prediction has to be done on-line and in real-time. The Elman recurrent net is thus chosen as the predictor. This is because among the available recurrent networks, it has the simplest architecture. In addition, it also has the characteristic of a dynamic memory. The simple architecture of the Elman net has the potential of real-time computation. The pseudo-impedance control rule is employed to train the predictor network to get fast convergence. The prediction is done based on the on-line learning and prediction scheme which allows predictor to carry on learning for each trial while making a prediction.

We have selected three particular motion series to test the predictive capability of the network predictor. The three motion series present different dynamic characteristics with certain nonlinearities. In the first simulation, the object in the circular motion keeps a constant speed but changing its moving direction with time. In the second simulation, the chair moves at a given direction, but its speed slowing down gradually due to the floor-friction force. As illustrated in Figure 5-3 and Figure 5-5, the prediction results of these two simulations shows the predictor quickly home in the trajectories of the object within only five sampling time after briefly training (i.e. 300 iterations).

For comparison purpose, we computed the normalized root mean square (NRMS) prediction errors of the two simulations, which are 6.87×10^{-2} and 7.93×10^{-2} respectively for x and y directions in first case, and 6.06×10^{-2} for x and y directions in second simulation. Therefore it is reasonable compared with 1.4×10^{-4} obtained by Lapedes and Farber (1987) who use backpropagation on 1000 sets of sample pairs. Note the NRMS error is defined as NRMS error = (RMS prediction error)/(standard deviation of the data).

In the final simulation, the chaotic time series presents significant nonlinearity in both direction and speed. As shown in Figure 5-8, the predictor still can manage to adapt its prediction even the chaotic time series presents drift changes with high nonlinearity. The simulation results seem to show that the predictor is capable of making predictions with reasonably accuracy in real-time. Furthermore, the inherent nonlinearity and learning capability enable the network predictor to predict adaptively even in the presence of high nonlinear, complicated "random" time series.

However, the performance of the prediction is less accurate than Lapedes and Farber's backpropagation method (i.e. 1.4×10^{-4}). This is because we use much less information (only four past samples) and much less training lessons (i.e. 300 sweeps) for predictions. The price paid in accuracy, however, is compensated by the speed and simplicity demanded by real-time and on-line requirements. On the other hand, the moving objects which may appear in the navigation tend to have "sensible" motions (i.e. they have fairly smooth trajectories), therefore the accuracy of the prediction we achieved seems to be reasonably acceptable.

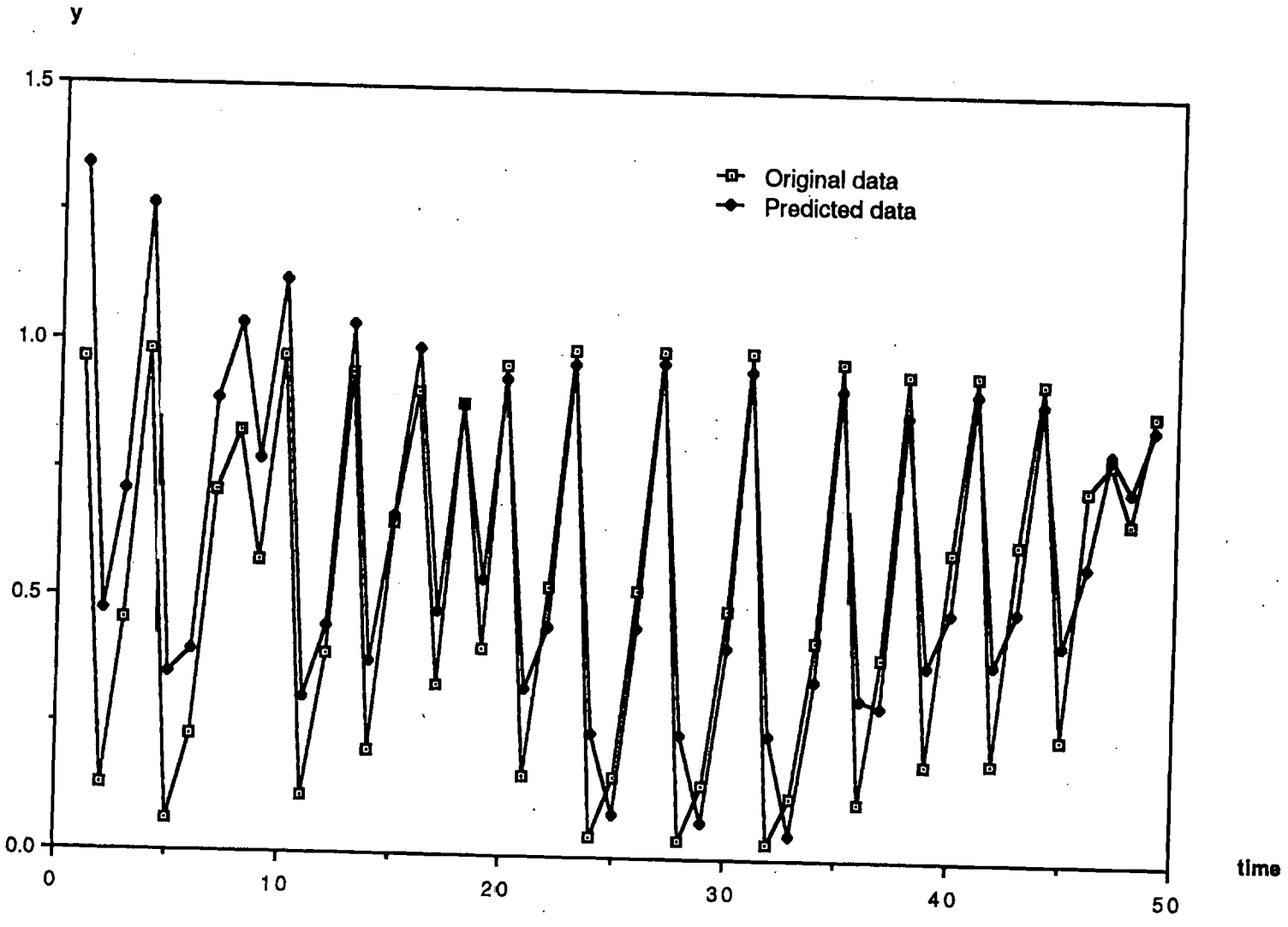


Figure 5-8: Prediction of a chaotic time-series

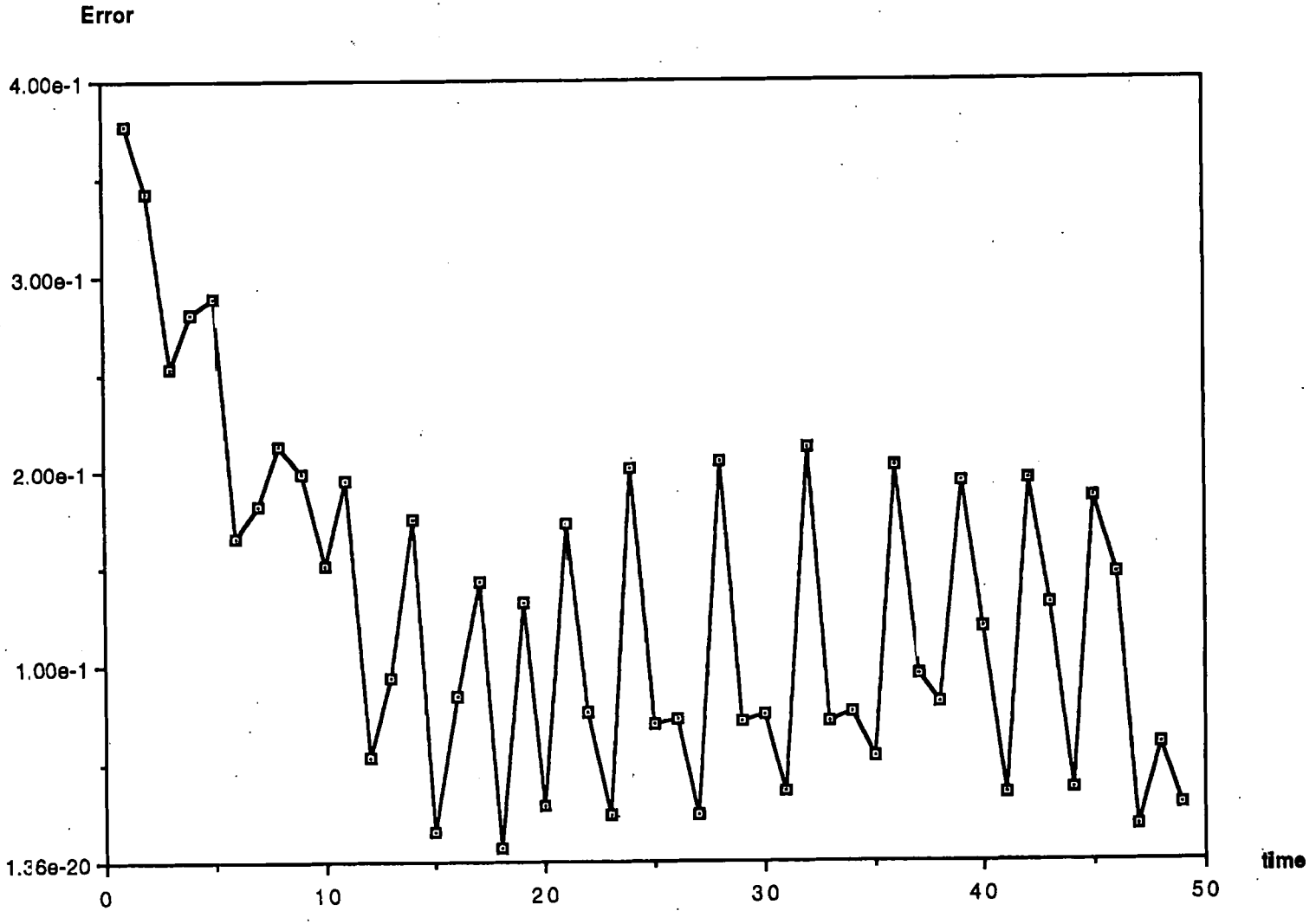


Figure 5-9. Prediction errors of a chaotic time-series

Chapter 6

Navigator-2 — An Improved Version of Navigator-1

This chapter focuses on the design and implementation of a navigator — Navigator-2 — which is an improved version of Navigator-1 described in chapter 4. The changes and revisions in this version of the navigator are mainly due to the introduction of a new factor to the navigation environment — allowing moving obstacles to be involved. With moving obstacles, the task of the navigator is much more difficult. This is because the geometric constraints of avoidance imposed by moving obstacles are time-varying. Moreover, the movements of the obstacles are unknown, so the steering decisions produced by the navigator are determined not only by the sensor readings but also by the estimation of the motions of the moving obstacles generated by the predictor (described in chapter 5).

In the following sections, we first present the main revisions needed to Navigator-1 in the presence of moving obstacles in the navigation environment. We then describe the structure of the Navigator-2 and its implementation.

6.1 Revising Navigator-1

In chapter 4, we described the Navigator-1, in which the navigation environment is simplified by assuming that only stationary obstacles exist. Clearly, this condition does not represent the real-world constraints. It is clear that a robot that can deal with moving obstacles will be capable of performing a much larger and more complex class of tasks. This thesis concerns the

navigation problem in time-varying environments where unpredictable events may happen to the navigation workspace. As we stated in the example in chapter 1, during a navigation, unexpected events which include moving obstacles with unknown trajectories and unknown stationary obstacles may happen in the workspace at any time. The navigator should be endowed with the capabilities of not only dealing with unknown static obstacles but also handling unexpected moving objects. Therefore, the function of the Navigator-1 needs to be extended. In this section, we present two main revisions to the Navigator-1.

a) Obstacle avoidance decomposition

When moving obstacles are present in the environment, the steering problem becomes much more complex than in the cases where only stationary obstacles are involved. This is because the geometric constraints of avoidance imposed by moving obstacles are time-varying. To avoid collisions with moving obstacles, it requires rather expensive computation since the safe positions of the robot in the workspace need to be computed successively.

In order to tackle the steering problem in the presence of time-varying conditions, as well as from the consideration that it is necessary to incorporate two other components (planner and predictor) for autonomous navigation, the idea of a forbidden region is introduced. A forbidden region is an area which is likely to be occupied by a moving object between a sampling time interval. It is constructed from the prediction about the movement of the object at the next sampling time. By creating a forbidden region at each sampling time for the moving object, the dynamic obstacle avoidance problem can be transferred into a static problem for each time increment, where the neural network based steering scheme proposed in section 4.3.1 of chapter 4 can then be employed directly. Since moving obstacle avoidance uses different

sources from that of the static obstacle avoidance, and the proposed steering scheme can be applied to them respectively, the complicated obstacle avoidance can thus be decomposed into two parallel and independent processes: static obstacle avoidance and dynamic obstacle avoidance.

Based on the decomposition, the revised navigator can be implemented by a structured network in which two sub-networks are used to realise dynamic obstacle avoidance and static obstacle avoidance respectively, and the third sub-net is used to make final steering decision by reconciling the results from those two sub-networks. By separating the process of moving obstacle avoidance from the process of static obstacle avoidance, the complexity of the steering problem is reduced significantly. This is because by considering moving obstacles and stationary obstacles independently, the possible patterns represented in terms of occupancy position or region in the immediate environment of the robot for each case become simple and enumerable. Thus, it is not difficult to construct two sets of training patterns to treat moving obstacles and stationary obstacles separately. In the alternative solution, one network trained to recognise both static and dynamic obstacles, it is difficult to construct a training set as the number of combinations of moving and static obstacles is enormous.

In addition, the use of a structured network allows both small size sub-networks to be trained independently. This reduces the number of training patterns for each of training processes and shortens the learning time, thus makes learning efficient. Furthermore, the small size of the sub-nets yields a basis for real-time performance which would make real-time navigation feasible.

b) The redefined steering commands in the steering scheme

The second major revision is to redefine the steering commands in the steering scheme. As described in section 4.3.1 of chapter 4, there were six steering commands which correspond to six moving actions (FW, BW, TL, TR, ST and GS). In representing turn actions, the commands turn left (TL), turn right (TR) were defined as 45° turns. Thus, to reach 3, or 2 in the local grid (see Figure 6-1) needs the robot to go back first and then turn to left or right respectively. This corresponds to the combination of the commands backward (BW) and TL, or BW and TR respectively.

In the situation where moving obstacles are present, however, the backward action of the robot might cause a collision with a moving object which is following behind the robot. This is because the local grid only provides the robot with information about the area to its front and sides. The moving object which follows behind the robot would not be detected. Therefore, the command backward is no longer used, unless otherwise stated. To cater for this new situation, we redefine the four moving actions: Forward left (FL), Forward right (FR), turn left (TL) and turn right (TR). With reference to Figure 6-1, to move the robot from MR to square 2 or 3 in the local grid requires command TR or TL. To reach square 4 or 5, needs command FR or FL.

13	14	6	7	8
12	5	1	4	9
11	3	MR	2	10

Figure 6-1. The local grid

This means that there are now a total of seven steering commands that are available. These are: – F(W), TL, TR, FL, FR, ST and GS.

In addition to the two major revisions described above, there are also some minor revisions which will be described while presenting the implementation of the Navigator-2. In the rest of this chapter, we first show the overall structure of the Navigator-2. We then present the detail of implementation of each sub-net in turn.

6.2. Structure of the Navigator-2

A block diagram of the Navigator-2 is shown in Figure 6-2. It consists of an AND combiner and three sub-networks: a static obstacle avoidance (SOA) sub-net, an dynamic obstacle avoidance (DOA) sub-net, and a decision-making sub-net.

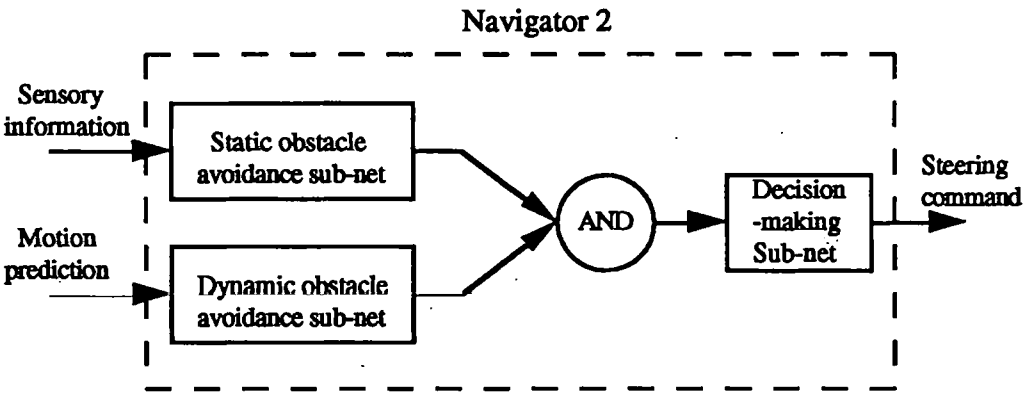


Figure 6-2. Structure of the Navigator-2

The neural network based steering scheme proposed in section 4.3.1 of chapter 4 is employed for both static obstacle avoidance and dynamic obstacle avoidance by means of two sub-nets. In the steering scheme, the steering commands are issued by directly converting the local information collected in the local grid through the network. In this way, the sensory and action information can be integrated directly so that sensor-based steering can be done in real-time in response to unexpected environmental changes. It can be seen from the figure that in the navigator the SOA sub-net uses the sensory information as input, and outputs a group of steering commands for avoiding

static obstacles. Meanwhile, the DOA sub-net produces a group of steering commands for avoiding moving obstacles based on the motion prediction. These two groups of action commands are then combined using a logical operation AND. Note that the number of outputs from each sub-net, corresponding to the action commands, is the same. The resultant commands from the AND operation satisfy the requirements of avoiding static obstacles as well as dynamic obstacles. They are then sent to the decision-making sub-net as inputs. The decision-making sub-net eventually generates the final collision-free action command.

Note that both input signals and output commands in each of sub-nets are represented by 1 bit each (i.e. binary value 0 or 1). If there are no moving (or stationary) obstacles, the DOA (or SOA) sub-net outputs 1 on all bits, which means no actions are constrained.

So far, we have described the overall structure of the Navigator-2. In the following sections, we describe the implementation details of each sub-net in turn.

6.3 Building Sub-net for Static Obstacle Avoidance

The static obstacle avoidance (SOA) sub-net is used to provide all possible steering commands for avoiding static obstacles by directly converting the local sensory information through a backpropagation network. The local grid is used to collect the local sensory information.

To ensure the existence of a feasible path for the robot in the presence of moving obstacles, the workspace with stationary obstacles is assumed to be less cluttered than that described in chapter 4. It is assumed that the minimum distance between any two static obstacles is greater than the width of the local

grid (i.e. 5 times the size of the square). With this assumption, only one static obstacle can be present in the local grid at each sampling time. This assures that the robot has enough space to avoid an unexpected moving obstacle among static obstacles, if there is one. Note that other definitions and assumptions used in this chapter are the same as those given in chapter 4.

In the following subsections, we show the architecture of the SOA sub-net and then present the training process.

6.3.1 Architecture of the static obstacle avoidance sub-net

The SOA sub-net consists of a three-layer backpropagation network, as shown in Figure 6-3. The input layer consists of 9 units. As illustrated in the figure, each square from the second-level of the local grid (square 6-14) is fed to the corresponding input unit. The coarse coding described in Navigator-1 is not used here. This is because if coarse coding was used, the sensory information would be compressed. The sub-net would then generate only a part of all the collision-free commands. This might cause the decision-making sub-net to fail to find a collision-free action command, even though one exists, since the final action is obtained by reconciling the combined results of the AND combiner from both the SOA sub-net and DOA sub-net.

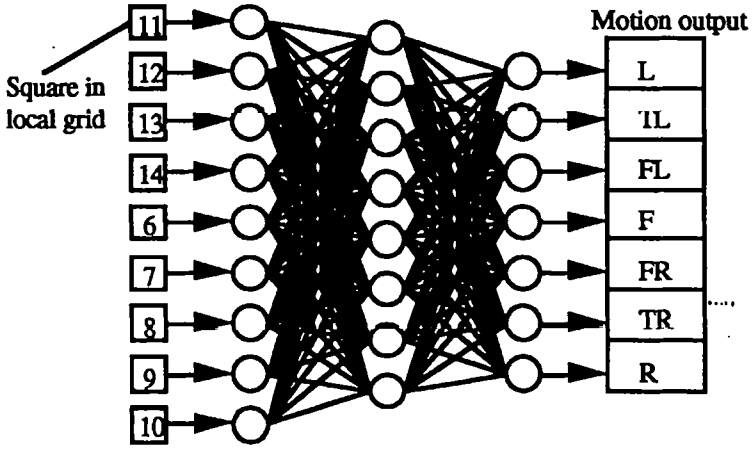


Figure 6-3. Architecture of the SOA sub-net

The output layer of the SOA sub-net consists of seven units which correspond to the five action commands: forward(F), forward right (FR), forward left (FL), turn right (TR) and turn left (TL); and two direction indicators: R and L. The direction indicators are used to indicate the obstacle-free direction among existing moving actions. When a group of collision-free action commands are obtained from the SOA and DOA sub-nets, the direction indicators help the decision-making sub-net to find the action with the highest priority to avoid obstacles while the robot moves towards a subgoal. The functions of the indicators will be illustrated in the next subsection. The number of hidden units in the SOA sub-net is empirically fixed to 8.

6.3.2 Training

To train the SOA sub-net, the network is presented with nine second-level squares (No. 6-14) in the local grid. The corresponding steering commands for avoiding a static obstacle and the obstacle-free direction are supplied as the desired output. The training patterns are constructed similarly to that described in section 4.3.3 of chapter 4 with a difference: here, the desired output consists of all possible collision-free steering commands rather than just one corresponding steering command. An example of generating a pair of training patterns is illustrated in Figure 6-4.

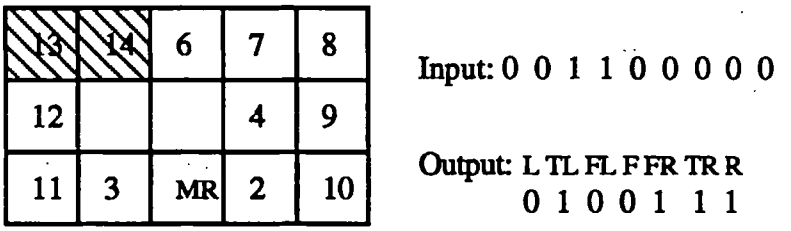


Figure 6-4. A example of a pair of training patterns

In Figure 6-4, the shaded squares represent the space occupied by an obstacle. The steering commands are TL, FR and TR which will direct the mobile robot

to the position 3, 4 or 2 respectively. The region on the right of the robot is the obstacle-free space, thus the direction indicator R has the value 1. Overall, 28 training patterns are generated in the same way as that described above and they are listed in Table 6-1. The number of training patterns is relatively small. One reason is that with the assumptions made at the beginning of section 6.3, only one static obstacle can be present in the local grid at one time. Additionally, only the second-level squares in the local grid are used to represent the occupation states of a static obstacle.

TABLE 6-1. Training patterns of the SOA sub-net

No	INPUT (square values)									OUTPUT						
	11	12	13	14	6	7	8	9	10	L	TL	FL	F	FR	TR	R
1	0	0	0	1	1	1	1	0	0	1	1	0	0	0	1	0
2	0	0	0	0	1	1	1	0	0	1	1	0	0	0	1	0
3	0	0	0	0	1	1	0	0	0	1	1	0	0	0	1	0
4	0	0	0	0	0	1	0	0	0	1	1	1	0	0	1	0
5	0	0	0	0	0	1	1	0	0	1	1	1	0	0	1	0
6	0	0	0	0	0	1	1	1	0	0	1	1	0	0	0	0
7	0	0	0	0	0	0	1	1	1	0	1	1	1	0	0	0
8	0	0	0	0	0	0	1	1	0	0	1	1	1	0	0	0
9	0	0	0	0	0	0	0	1	0	0	1	1	1	0	0	0
10	0	0	0	0	0	0	0	1	1	0	1	1	1	0	0	0
11	0	0	0	0	0	0	1	0	0	1	1	1	1	0	1	0
12	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0
13	0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1
14	0	0	0	1	1	1	0	0	0	1	1	0	0	0	1	1
15	0	0	1	1	1	1	1	0	0	1	1	0	0	0	1	1
16	0	0	1	1	1	1	0	0	0	0	1	0	0	0	1	1
17	0	0	1	1	1	0	0	0	0	0	1	0	0	0	1	1
18	0	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1
19	0	0	0	1	0	0	0	0	0	0	1	0	0	1	1	1
20	0	0	1	1	0	0	0	0	0	0	1	0	0	1	1	1
21	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	0
22	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0
23	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0
24	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0
25	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0
26	0	0	1	0	0	0	0	0	0	0	1	0	1	1	1	1
27	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
28	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1

In chapter 4, the navigator network is trained with all the possible training patterns which is not an efficient approach. The network based navigator

should be able to determine its action in a new situation by inferring from the necessary patterns it has learned. In addition, in the training experiments we conducted, we found that if all the patterns are present, some of the training patterns would make the training process difficult to converge.

To overcome these problems, a more efficient training method is developed. In this method, only necessary patterns are taught to the SOA sub-net rather than over all possible patterns. The SOA sub-net determines its steering commands in any environments by inferring from the necessary patterns it has learnt. The necessary patterns are chosen by multi-phase training and testing. In the first phase of training and testing, the SOA sub-net is trained briefly (i.e. 2000 iterations) with a group of training patterns which are randomly chosen from Table 6-1. The trained network is tested by examining the results of given inputs (known or unknown). If the trained network presents faulty outputs for an unknown or known input, a training pattern which can produce the desired output for that input is added to the group of patterns, regardless of whether this pattern was in the group before or not. In the second phase, the SOA sub-net is re-trained using the updated training patterns, and then tested again. This process is repeated until no faulty outputs are found. In this way, the necessary training patterns are obtained, and then used to train the network to get convergence satisfactorily. This also prevents overtraining, as described by Hecht-Nielsen (1989). These patterns are listed in Table 6-2. Note that since the output of the network is represented by binary values, the faulty output is defined as less (or more) than 0.5 if the desired value is 1 (or 0).

During training, the standard backpropagation learning algorithm with a momentum term is used to adjust the weights, and the sigmoid function is used as the transfer function. Training is achieved by using the simulation package.

TABLE 6-2. The necessary training patterns of the SOA sub-net

No	INPUT (square values)									OUTPUT						
	11	12	13	14	6	7	8	9	10	L	TL	FL	F	FR	TR	R
1	0	0	0	0	1	1	1	0	0	1	1	0	0	0	1	0
2	0	0	0	0	1	1	0	0	0	1	1	0	0	0	1	0
3	0	0	0	0	0	1	0	0	0	1	1	1	0	0	1	0
4	0	0	0	0	0	1	1	0	0	1	1	1	0	0	1	0
5	0	0	0	0	0	1	1	1	0	0	1	1	0	0	0	0
6	0	0	0	0	0	0	1	1	0	0	1	1	1	0	0	0
7	0	0	0	0	0	0	0	1	0	0	1	1	1	0	0	0
8	0	0	0	0	0	0	0	1	1	0	1	1	1	0	0	0
9	0	0	0	0	0	0	1	0	0	1	1	1	1	0	1	0
10	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0
11	0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1
12	0	0	0	1	1	1	0	0	0	1	1	0	0	0	1	1
13	0	0	1	1	1	1	1	0	0	1	1	0	0	0	1	1
14	0	0	1	1	1	0	0	0	0	0	1	0	0	0	1	1
15	0	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1
16	0	0	0	1	0	0	0	0	0	0	1	0	0	1	1	1
17	0	0	1	1	0	0	0	0	0	0	1	0	0	1	1	1
18	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	0
19	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0
20	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0
21	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0
22	0	0	1	0	0	0	0	0	0	0	1	0	1	1	1	1
23	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
24	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
25	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1

At each training phase, the momentum term is set to 0.5, the learning rate of the hidden layer and output layer is set to 0.4. and 0.2 respectively. When the necessary training patterns are obtained with 2000 sweeps, 1000-sweep further training is added to get the network satisfactorily converges to the desired patterns in Table 6-1 with the RMS error 0.096.

The SOA sub-net is thus derived with learned connection weights as shown in Table 6-3 (a), (b). When an input pattern collected from the local grid is received, the trained SOA sub-net generates an output vector in which the continuous value of each bit equals 1 or 0 approximately and corresponds to a specific steering command or a moving direction.

TABLE 6-3(a). Hidden layer connection weights of the SOA sub-net

Input Layer	Hidden Layer							
	Node1	Node2	Node3	Node4	Node5	Node6	Node7	Node8
Bias	1.5939	-0.5586	1.1629	-1.7562	0.2770	1.1980	0.9333	-1.7508
Node1	-2.0002	1.0880	1.3923	-0.7970	-2.0148	1.6442	-1.6683	-1.8872
Node2	-4.9726	2.8405	1.3866	3.6892	-2.9307	1.0695	0.6982	-0.9897
Node3	-0.0150	2.2366	1.7074	1.6550	-0.5779	-0.4828	-1.1559	0.0379
Node4	0.4457	2.2898	2.5392	2.5460	-1.3195	-2.6726	-2.3183	1.2363
Node5	2.4953	0.4542	0.0258	2.4671	1.4758	-3.5251	-0.3614	3.7585
Node6	-0.2374	-2.5946	-2.3226	2.9433	2.2083	-0.4809	1.8039	2.9506
Node7	-0.6104	-1.8922	-1.5243	1.7903	1.7826	0.6095	1.0119	1.2609
Node8	-3.6350	-2.3778	-3.1455	0.5567	2.6051	1.6936	-5.1322	-1.7652
Node9	-2.7541	-2.0319	-1.6235	-1.2080	0.8611	1.3956	1.3238	-1.7381

TABLE 6-3(b). Output layer connection weights of the SOA sub-net

Hidden Layer	Output Layer						
	Node1	Node2	Node3	Node4	Node5	Node6	Node7
Bias	-1.7283	0.2509	0.9570	1.0773	0.6089	-0.3464	-1.0234
Node1	2.6839	3.4612	-0.7698	-0.7890	-0.4239	2.9302	4.6460
Node2	-2.4355	-1.9509	-3.2183	0.1443	2.2988	2.4616	1.4016
Node3	-0.8909	-1.4657	-1.3110	0.6287	3.0494	2.3779	2.4117
Node4	-1.6832	0.8239	-3.4917	-3.7254	-2.2362	1.3767	-3.5712
Node5	1.2249	3.3934	2.3231	0.0516	-3.2772	-2.1040	-0.3523
Node6	-1.5416	-1.2938	3.6932	3.4859	0.7672	-1.9817	-1.4489
Node7	4.5753	0.5625	1.0902	1.3605	1.6542	0.5941	-2.1196
Node8	0.9000	2.3427	-1.7665	-4.0756	-3.7113	1.9869	0.3248

6.4 Building the Sub-net for Dynamic Obstacle Avoidance

The dynamic obstacle avoidance (DOA) sub-net is used to provide all possible steering commands for avoiding moving obstacles based on the current motion prediction of a disruptive obstacle. The task of dynamic obstacle avoidance is much more difficult than that of static obstacle avoidance since moving obstacles impose time-varying constraints on the motion of the robot. The concept of a forbidden region here is employed to represent these time-varying constraints. By creating a forbidden region based on the prediction results generated by the predictor, the problem of moving obstacle avoidance can be transferred into a problem of static obstacle avoidance in which the forbidden region is treated as a 'virtual' static obstacle at each sampling time. The created forbidden region may overlap fully or partially with the *local grid*

which we defined in section 4.3.1 of chapter 4. The information about the forbidden region can then be collected as inputs to the DOA sub-net by means of the local grid. All possible steering commands for moving obstacle avoidance are thus generated by directly converting the 'virtual sensory information' from the local grid through the network.

In the following subsections, we first describe how to create a forbidden region for a moving obstacle based on its motion prediction. We then present the architecture of the DOA sub-net. Finally, we show the training process of the DOA sub-net.

6.4.1 Constructing the forbidden regions

During a navigation, the predictor keeps tracking the moving obstacle which is regarded as disruptive. When the predicted position of the moving obstacle is approximately at or near the position of the second-level square in the local grid (i.e. the distance between the predicted position of the moving obstacle and the position of the robot approximately less than or equal to a predefined distance, 2.828h, as shown in Figure 6-5. Here h is the size of the square in the local grid), the process of constructing forbidden regions starts, based on the prediction at each sampling time. The forbidden regions are created successively until the moving obstacle moves sufficiently far from the robot (i.e. greater than a predefined safe distance 3h).

The forbidden region at a time is determined in terms of the predicted position, the predicted moving direction and the current position of the disruptive moving obstacle. It is constructed by firstly connecting the current position to the predicted position, and then extending the area along the predicted moving direction to a predefined distance. Note that the predefined distance is in a direct ratio with the speed of the moving object. Here, it is

assigned to the distance between the current position and the predicted next position of the moving object. An example of creating a forbidden region is illustrated in Figure 6-5, where C is the current position, 13 is the predicted next position of the moving object, and the forbidden region consists of the shaded cells (5, 13 and C). It can be seen that the forbidden region overlaps not only with some of the second-level squares (No. 6-14) but also with some of the first-level squares (No. 1-5) in the local grid. This allows the attitude of the moving object to be represented in the local grid, and more attention can then be paid to dynamic obstacle avoidance when the information collected by the local grid is used as the input.

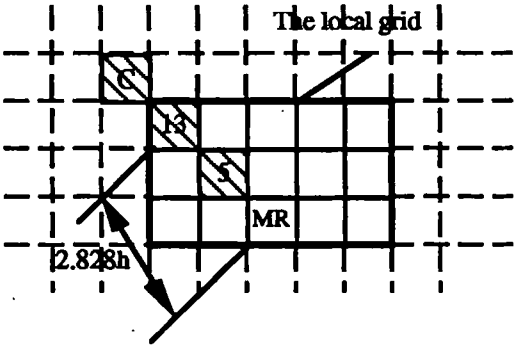


Figure 6-5. A example of creating a forbidden region

Note that a non-binary value (e.g. 5 as used in this solution, but could be any other positive value greater than 1) is assigned to each cell in the forbidden region to represent its 'virtual' occupation state. There are totally 78 cases in terms of the predicted position and the current position, and the corresponding forbidden region created for each case is listed in Appendix A.

6.4.2 Architecture of the dynamic obstacle avoidance sub-net

The architecture of the DOA sub-net consists of a single hidden layer backpropagation network as shown in Figure 6-6. The input layer consists of 13 units. The squares in the local grid are fed to the corresponding input units

respectively. The local grid is used to collect the 'virtual' sensory information where the squares overlapping with the forbidden region have non-binary values to represent the 'virtual' occupation state at that time, and the non-overlapping squares have the assigned value 0.

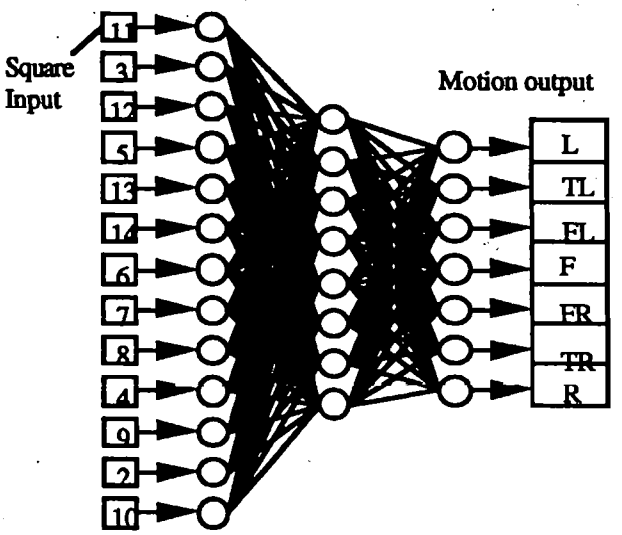


Figure 6-6. Architecture of the DOA sub-net

Identical to that of the SOA sub-net, the output layer of the DOA sub-net also consists of seven units which correspond to the five action commands: forward(F), forward right (FR), forward left (FL), turn right (TR) and turn left (TL); and two direction indicators: R and L. The number of hidden units in the DOA sub-net is empirically fixed to 8.

6.4.3 Training

To train the DOA sub-net, the network is presented with the squares in the local grid which overlap partially or fully with the forbidden region of a moving obstacle. The overlapping squares thus have the assigned non-binary values to represent the 'virtual' occupancy status. When the non-overlapping squares in the local grid are fed to the input units, their values are set to 0, regardless whether their real values are 1 or 0. Note that the value "1" means

that the square is actually occupied, otherwise 0. The corresponding overall steering commands for avoiding moving obstacles and the obstacle-free direction are supplied as the desired output. The set of training patterns are constructed by enumerating all the possible situations in terms of what kinds of 'virtual' occupancy patterns are in the local grid, and their corresponding commands. Figure 6-7 illustrates a example for generating a pair of training patterns.

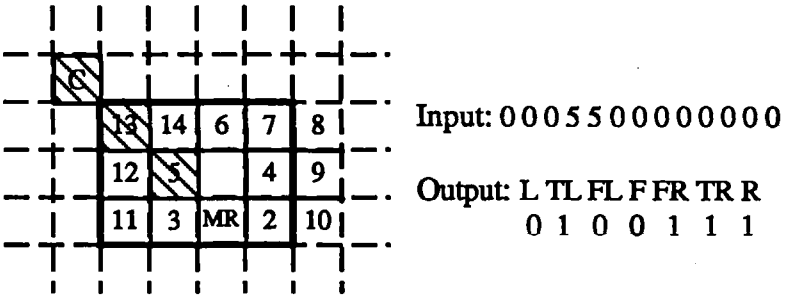


Figure 6-7. A example of a pair of training patterns

where shaded squares represent the forbidden region of the moving obstacle, the position 13 and 5 in the local grid is the overlapping area that has the assigned non-binary value 5. The overlapping squares are treated as a 'virtual' static obstacle at that time. Therefore, the overall collision-free action commands are TL, FR and TR which direct the robot to the position 3, 4 and 2 respectively. The region on the right of the robot is the 'obstacle-free' space, the direction indicator R thus has the value 1. Note that it is assumed that there is at most one moving obstacle within the predefined distance (2.818h) from the robot at a time. This means only one forbidden region may need to be constructed at sampling time. The 'virtual' occupied patterns represented by the local grid are therefore enumerable. Table 6-4 lists 55 training patterns based on the different cases of the forbidden region given in Appendix A. Note that for the purpose of simplicity, we use binary training patterns to train the network as in the Table 6-4, when a non-binary value is collected by the local grid, it is first divided by an integer 5, then sent to the DOA sub-net as the input.

TABLE 6-4. Training patterns of the DOA sub-net

No	INPUT (square values)													OUTPUT						
	11	3	12	5	13	14	6	7	8	4	9	2	10	L	TL	FL	F	FR	TR	R
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
3	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
4	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
5	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
6	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
7	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0
8	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	1	1	1
9	0	0	1	0	1	1	0	0	0	0	0	0	0	0	1	0	0	1	1	1
10	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1
11	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1	0	0	1	1	1
12	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	1	1	1
13	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	1	1
14	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1	0	0	1	1	1
15	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1
16	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
17	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0
18	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0
19	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
20	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
21	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0
22	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0
23	0	0	0	1	1	1	1	0	0	1	0	0	0	0	1	0	0	0	1	1
24	0	0	0	1	0	1	1	1	0	0	0	0	0	0	1	0	0	0	1	1
25	1	1	1	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	1	0
26	0	1	0	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	1	0
27	0	0	0	1	0	1	1	1	0	1	0	0	0	1	1	0	0	0	1	1
28	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	0	0	0	1	1
29	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	1	0	0	0
31	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	1	1	0	0	0
32	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	1	0	0	0
33	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	1	0	0	0
34	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	1	1	0	0	0
35	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	1	1	0	0	0
36	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	0	0	1	0
37	0	0	0	0	0	0	0	1	1	0	1	0	0	1	1	1	0	0	1	0
38	0	0	0	0	0	0	0	0	1	1	0	0	0	1	1	1	0	0	1	0
39	0	0	0	0	0	0	1	1	1	0	0	0	0	1	1	1	0	0	1	0
40	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1	1	0	0	1	0
41	0	0	0	0	0	0	0	1	0	0	1	0	0	1	1	1	0	0	1	0
42	0	0	0	0	0	0	1	1	1	1	0	0	0	1	1	1	0	0	1	0
43	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	1	0	1	0
44	0	0	0	0	0	0	0	0	1	1	1	0	1	0	1	1	0	0	0	0
45	0	0	0	0	0	0	0	1	0	1	1	0	1	0	1	1	0	0	0	0
46	0	0	0	0	0	0	1	1	0	1	1	0	0	0	1	1	0	0	0	0
47	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1	1	0	0	0	0
48	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	0	0
49	0	0	0	0	0	0	0	1	0	1	1	1	0	0	1	1	0	0	0	0
50	0	0	0	0	0	0	0	1	1	1	1	1	0	0	1	1	0	0	0	0
51	0	0	0	1	0	0	1	1	1	1	0	0	0	1	1	0	0	0	1	0
52	0	0	0	0	0	1	1	1	0	1	0	0	0	1	1	0	0	0	1	0
53	0	0	0	1	0	0	1	1	0	1	1	1	1	0	1	0	0	0	0	0
54	0	0	0	1	0	1	1	0	0	1	0	1	0	0	1	0	0	0	0	0
55	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1

TABLE 6-5. The necessary training patterns of the DOA sub-net

No	INPUT (square values)													OUTPUT						
	11	3	12	5	13	14	6	7	8	4	9	2	10	L	TL	FL	F	FR	TR	R
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
3	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
4	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
5	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
6	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
7	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0
8	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	1	1	1
9	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1
10	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1	0	0	1	1	1
11	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	1	1	1
12	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	1	1
13	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1
14	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
15	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0
16	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0
17	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
18	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0
19	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0
20	0	0	0	1	1	1	1	0	0	1	0	0	0	0	1	0	0	0	1	1
21	0	0	0	1	0	1	1	1	0	0	0	0	0	0	1	0	0	0	1	1
22	1	1	1	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	1	0
23	0	1	0	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	1	0
24	0	0	0	1	0	1	1	1	0	1	0	0	0	1	1	0	0	0	1	1
25	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	0	0	0	1	1
26	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	1	0	0	0
28	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	1	1	0	0	0
29	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	1	0	0	0
30	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	1	0	0	0
31	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	1	1	0	0	0
32	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	1	1	0	0	0
33	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	0	0	1	0
34	0	0	0	0	0	0	0	0	1	1	0	0	0	1	1	1	0	0	1	0
35	0	0	0	0	0	0	1	1	1	0	0	0	0	1	1	1	0	0	1	0
36	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1	1	0	0	1	0
37	0	0	0	0	0	0	0	1	0	0	1	0	0	1	1	1	0	0	1	0
38	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	1	0	1	0
39	0	0	0	0	0	0	0	0	1	1	1	0	1	0	1	1	0	0	0	0
40	0	0	0	0	0	0	0	1	0	1	1	0	1	0	1	1	0	0	0	0
41	0	0	0	0	0	0	1	1	0	1	1	0	0	0	1	1	0	0	0	0
42	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1	1	0	0	0	0
43	0	0	0	0	0	0	0	1	0	1	1	1	0	0	1	1	0	0	0	0
44	0	0	0	0	0	0	0	1	1	1	1	1	0	0	1	1	0	0	0	0
45	0	0	0	1	0	0	1	1	1	1	0	0	0	1	1	0	0	0	1	0
46	0	0	0	0	0	1	1	1	0	1	0	0	0	1	1	0	0	0	1	0
47	0	0	0	1	0	0	1	1	0	1	1	1	1	0	1	0	0	0	0	0
48	0	0	0	1	0	1	1	0	0	1	0	1	0	0	1	0	0	0	0	0
49	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
50	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1

The training method presented in subsection 6.3.2 is used here to train the

DOA sub-net, in which only necessary patterns are presented to the network. The necessary training patterns obtained through the multi-phase of training and testing are listed in Table 6-5.

TABLE 6-6(a). Hidden layer connection weights of the DOA sub-net

Input Layer	Hidden Layer							
	Node1	Node2	Node3	Node4	Node5	Node6	Node7	Node8
Bias	2.2879	-2.6891	-2.8784	-0.8185	-1.7259	0.6987	-2.2470	1.5348
Node1	-1.4332	-0.6762	-0.4796	2.8722	3.5209	-2.5379	-0.7995	-0.5776
Node2	-1.2467	0.3283	0.8789	2.2695	2.6459	-3.0008	1.7811	-1.1092
Node3	-0.6992	-0.3879	-0.5164	1.7746	2.4839	-2.5558	-0.0473	-0.7691
Node4	-1.5967	-1.5661	0.6628	2.0258	1.7493	-0.2972	2.5113	-1.1594
Node5	-1.2128	-1.2472	-1.3620	2.8947	-0.8432	1.7159	1.8214	-1.2828
Node6	-2.8084	-1.1199	0.9300	0.6243	-2.1250	2.3779	2.4667	-0.4087
Node7	-1.1201	1.4884	1.0657	0.5760	1.1638	-0.5691	-0.1393	-0.5991
Node8	-0.4083	-0.9631	3.2908	-0.8595	-1.1405	-0.3486	2.5394	0.7310
Node9	0.5200	-0.5716	3.1079	-1.7683	-0.9830	-1.9559	0.3654	0.5360
Node10	-1.3204	2.4329	2.8219	-2.5876	-0.6885	-0.0146	0.2189	-0.1947
Node11	0.9030	2.5499	0.1204	-1.1300	0.9227	-0.2248	-0.3386	-2.1022
Node12	-1.3360	3.3762	1.8083	-0.8909	0.1537	0.8689	-0.6220	-1.9918
Node13	-0.0711	4.3719	-0.9558	-1.4026	0.1865	0.4859	-1.4367	-2.1810

TABLE 6-6(b). Output layer connection weights of the DOA sub-net

Hidden Layer	Output Layer						
	Node1	Node2	Node3	Node4	Node5	Node6	Node7
Bias	-0.5747	0.5823	-0.7206	0.1956	-0.0069	0.6311	-0.8927
Node1	1.3662	1.4981	4.2252	3.4206	0.0506	0.4722	-0.4546
Node2	-4.2644	1.0765	1.4004	0.3392	-3.2604	-4.5155	-4.1372
Node3	1.0395	0.9671	0.6824	-5.1735	-5.3843	-0.4417	-2.0945
Node4	-4.2326	-2.0264	-3.6710	0.0760	2.3765	1.4547	1.2071
Node5	-1.7497	-3.4879	-1.8930	2.0649	1.5269	0.9904	-3.6073
Node6	-0.5724	4.0789	-0.4120	-0.3788	0.3176	-1.1998	4.2042
Node7	0.9384	0.1078	-3.1300	-5.0687	-0.2362	2.4247	0.0935
Node8	3.4533	0.6266	1.1752	1.4059	1.2203	2.9055	0.4362

In training, the learning rule and transfer function are the same as that used in training the SOA sub-net, so are the procedure of training and the learning parameters. The DOA sub-net finally converges to the desired patterns with RMS error 0.0827. The DOA sub-net is thus derived with learned connection

weights as shown in Table 6-6 (a), (b).

6.5 Building Sub-net for Decision-making

In this section, we present the decision-making sub-net. Once two groups of action commands are returned from the SOA sub-net and the DOA sub-net in parallel, they are firstly combined by a logical combiner – AND. This is to get such action commands that meet the collision-free requirements of avoiding both moving and stationary obstacles. The resultant action commands are then sent to the decision-making sub-net. The task of the decision-making sub-net is to establish control priorities among resultant action commands, and thus determine the final steering action to avoid collisions with both moving and static obstacles while directing robot towards the current subgoal.

Before we introduce the architecture of the decision-making sub-net and the training process, we first describe the function of the AND combiner briefly.

6.5.1 AND combiner

The AND combiner is used to get a common set of commands and direction indicator between the two groups of action commands generated by the SOA and DOA sub-net respectively. By using the AND combiner, only a combination of two groups of commands is used as the input of the decision-making sub-net rather than two group of steering commands. This would reduce the number of input units in the decision-making sub-net, and simplify the training patterns. Thus the complexity of implementing the decision-making sub-net is reduced sufficiently.

The AND function is an essential component of the whole decision-making process. Only commands that are common to both the DOA and the SOA sub-

nets are to be acted upon. If there is a difference of opinion between the two sub-nets, the opinions should be ignored. In the extreme case where the sub-nets have no common opinions, the result of the AND is all zeros, which corresponds to the command STOP. Since this is a well-defined function, there is little to be gained by incorporate the AND function into the decision making network. It was therefore decided to make the two operations – the AND function and the decision-making network – separate.

Using two groups of commands as the two inputs, each of which is represented as a 7-bit binary vector, the function of the AND combiner can be described as follows: a bit of the output vector is 1 if both corresponding bits of two inputs are 1, otherwise that bit of the output vector is 0. Note that each bit of an input and output vector corresponds to a specific command or a direction indicator.

6.5.2 Architecture of the decision-making sub-net

The decision-making sub-net consists of a three-layer backpropagation network, as shown in Figure 6-8.

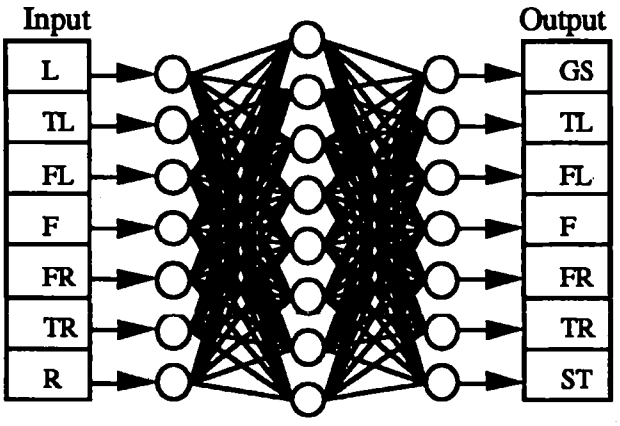


Figure 6-8. Architecture of the decision-making sub-net

The input layer comprises 7 units which correspond to the output of the AND combiner (i.e. the five action commands: F, FR, FL, TR and TL and two

direction indicators: R and L). The output layer also consists of 7 units which correspond to the seven steering commands: go subgoal (GS), turn left (TL), turn right (TR), forward left (FL), forward right (FR), forward (F) and stop (ST). If there are no obstacles across the path, i.e. if the output vector of both SOA and DOA sub-net are 1, then the robot just goes towards the subgoal. In this situation, the command 'go subgoal' is used. If the resultant command vector from the AND combiner is 0, it means that no collision-free actions could be found. In this case, the command 'stop' is used. The number of hidden units in the decision-making sub-net is empirically fixed to 8.

6.5.3 Training

To train the decision-making sub-net, the network is presented with the resultant action commands and direction indicator from the AND combiner. The command with the highest priority is supplied as the desired output. The control priority among the resultant action commands is established as follows:

The commands can be divided into two sets of actions. One set is FL, TL and L, the other is FR, TR and R.

a) If there is a direction indicator L (or R) among the resultant commands, the commands which lead the robot to the L (or R) side in the local grid such as FL, TL and have higher priority.

b) Within a set of commands, the FL (or FR) has higher priority than the TL (or TR).

c) The command F has higher priority than the TL and the TR, but the same priority with FL and FR.

Based on the priorities of the action commands, 37 training patterns are generated, which are shown in Table 6-7.

TABLE 6-7. Training patterns of decision-making sub-net

No	INPUT							OUTPUT						
	L	TL	FL	F	FR	TR	R	GS	TL	FL	F	FR	TR	ST
1	0	1	0	0	0	0	0	0	1	0	0	0	0	0
2	1	1	0	0	0	0	0	0	1	0	0	0	0	0
3	1	1	0	0	0	1	0	0	1	0	0	0	0	0
4	0	0	1	0	0	0	0	0	0	1	0	0	0	0
5	0	0	1	0	0	1	0	0	0	1	0	0	0	0
6	0	0	1	1	0	1	0	0	0	1	0	0	0	0
7	0	1	1	0	0	0	0	0	0	1	0	0	0	0
8	1	1	1	0	0	0	0	0	0	1	0	0	0	0
9	0	1	1	1	0	0	0	0	0	1	1	0	0	0
10	1	1	1	1	0	0	0	0	0	1	1	0	0	0
11	1	1	1	0	0	1	0	0	0	1	0	0	0	0
12	1	1	1	1	0	1	0	0	0	1	1	0	0	0
13	0	0	1	1	0	0	0	0	0	1	1	0	0	0
14	1	1	1	1	1	0	0	0	0	1	1	0	0	0
15	0	1	0	0	0	1	0	0	1	0	0	0	1	0
16	1	1	0	0	0	1	1	0	1	0	0	0	1	0
17	0	0	0	0	0	1	0	0	0	0	0	0	1	0
18	0	0	0	0	0	1	1	0	0	0	0	0	1	0
19	0	1	0	0	0	1	1	0	0	0	0	0	1	0
20	0	0	0	0	1	0	0	0	0	0	0	1	0	0
21	0	1	0	0	1	0	0	0	0	0	0	1	0	0
22	0	1	0	1	1	0	0	0	0	0	0	1	0	0
23	0	0	0	0	1	1	0	0	0	0	0	1	0	0
24	0	0	0	0	1	1	1	0	0	0	0	1	0	0
25	0	0	0	1	1	1	0	0	0	0	1	1	0	0
26	0	0	0	1	1	1	1	0	0	0	1	1	0	0
27	0	1	0	0	1	1	1	0	0	0	0	1	0	0
28	0	1	0	1	1	1	1	0	0	0	1	1	0	0
29	0	0	0	1	1	0	0	0	0	0	1	1	0	0
30	0	0	1	1	1	1	1	0	0	0	1	1	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	1
32	0	0	1	1	1	0	0	0	0	0	1	0	0	0
33	0	1	0	1	0	1	0	0	0	0	1	0	0	0
34	0	1	0	1	0	0	0	0	0	0	1	0	0	0
35	0	0	0	1	0	1	0	0	0	0	1	0	0	0
36	0	0	0	1	0	0	0	0	0	0	1	0	0	0
37	1	1	1	1	1	1	1	1	0	0	0	0	0	0

During training, the efficient training method described in subsection 6.3.2 is also used. The necessary patterns obtained through the multi-phase training and testing are listed in Table 6-8. The learning rule and the transfer function are the same as that used in training the SOA sub-net.

TABLE 6-8. The necessary training patterns of the decision-making sub-net

No	INPUT							OUTPUT						
	L	TL	FL	F	FR	TR	R	GS	TL	FL	F	FR	TR	ST
1	0	1	0	0	0	0	0	0	1	0	0	0	0	0
2	1	1	0	0	0	0	0	0	1	0	0	0	0	0
3	1	1	0	0	0	1	0	0	1	0	0	0	0	0
4	0	0	1	0	0	0	0	0	0	1	0	0	0	0
5	0	0	1	0	0	1	0	0	0	1	0	0	0	0
6	0	0	1	1	0	1	0	0	0	1	1	0	0	0
7	0	1	1	0	0	0	0	0	0	1	0	0	0	0
8	1	1	1	0	0	0	0	0	0	1	0	0	0	0
9	0	1	1	1	0	0	0	0	0	1	1	0	0	0
10	0	0	1	1	0	0	0	0	0	1	1	0	0	0
11	1	1	1	1	1	0	0	0	0	1	1	0	0	0
12	0	1	0	0	0	1	0	0	1	0	0	0	1	0
13	1	1	0	0	0	1	1	0	1	0	0	0	1	0
14	0	0	0	0	0	1	0	0	0	0	0	0	1	0
15	0	0	0	0	0	1	1	0	0	0	0	0	1	0
16	0	1	0	0	0	1	1	0	0	0	0	0	1	0
17	0	0	0	0	1	0	0	0	0	0	0	1	0	0
18	0	1	0	0	1	0	0	0	0	0	0	1	0	0
19	0	1	0	1	1	0	0	0	0	0	1	1	0	0
20	0	0	0	0	1	1	0	0	0	0	0	1	0	0
21	0	0	0	0	1	1	1	0	0	0	0	1	0	0
22	0	0	0	1	1	1	0	0	0	0	1	1	0	0
23	0	0	0	1	1	0	0	0	0	0	1	1	0	0
24	0	0	1	1	1	1	1	0	0	0	1	1	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	1
26	0	0	0	0	0	0	0	0	0	0	0	0	0	1
27	0	0	0	0	0	0	0	0	0	0	0	0	0	1
28	0	0	1	1	1	0	0	0	0	0	1	0	0	0
29	0	1	0	1	0	1	0	0	0	0	1	0	0	0
30	0	1	0	1	0	0	0	0	0	0	1	0	0	0
31	0	0	0	1	0	1	0	0	0	0	1	0	0	0
32	0	0	0	1	0	0	0	0	0	0	1	0	0	0
33	1	1	1	1	1	1	1	1	0	0	0	0	0	0
34	1	1	1	1	1	1	1	1	0	0	0	0	0	0

TABLE 6-9(a). Hidden layer connection weights of the decision-making sub-net

Input Layer	Hidden Layer							
	Node1	Node2	Node3	Node4	Node5	Node6	Node7	Node8
Bias	0.9348	-0.2213	-1.6384	0.7476	-1.1432	1.2903	-0.3770	-0.7003
Node1	-3.4849	-1.4195	-0.2067	-2.1452	0.3767	-0.1459	1.8536	2.0261
Node2	-1.8512	-0.4464	-0.2270	-1.7199	2.2684	-0.0971	-0.1209	2.5089
Node3	-3.0142	-0.8883	3.5436	-3.6521	-4.9414	3.6908	2.8777	1.9016
Node4	-0.3648	-1.0712	2.3357	4.0043	-3.5084	-4.0305	2.2957	-2.1197
Node5	3.0357	-2.9208	2.2563	3.0187	3.8352	-4.1889	-3.4754	-0.5311
Node6	-0.6647	1.8645	0.9445	-2.0887	1.9208	0.8262	-0.9458	-1.3623
Node7	1.8003	1.0576	1.1263	-0.3183	1.2435	-0.5945	-3.8828	-2.3344

TABLE 6-9(b). Output layer connection weights of the decision-making sub-net

Hidden Layer	Output Layer						
	Node1	Node2	Node3	Node4	Node5	Node6	Node7
Bias	0.2357	-1.0759	-1.8753	-0.0349	-1.7805	-1.0354	-0.9263
Node1	-2.6126	-3.3772	-3.2233	-1.0537	3.3900	-0.4751	1.5348
Node2	-1.8547	-0.1012	-0.9846	-0.1991	-2.4693	3.2798	0.0253
Node3	2.1725	-3.7227	1.0074	-0.3664	-0.1009	-2.1838	-3.4079
Node4	-2.3506	-1.9518	-2.0305	3.8467	1.3774	-2.5089	0.9816
Node5	0.0416	2.8269	-5.2879	-4.7895	1.6900	1.7184	-1.9757
Node6	-1.9223	0.3957	1.7566	-4.3994	-3.3559	1.6158	1.5143
Node7	-2.3924	-1.4446	3.0548	2.4493	-3.9316	-3.7992	-0.3007
Node8	-0.6988	3.1610	0.8158	-1.8840	-1.8034	-2.3822	-0.3198

At each training phase, the momentum term is set to 0.5, the learning rate of the hidden layer and output layer are set to 0.4. and 0.3 respectively. After 3788 sweeps, the decision-making sub-net converges to the desired patterns in Table 6-7 with RMS error 0.0783. The decision-making sub-net is derived with learned connection weights as shown in Table 6-9 (a), (b).

6.6 Summary

In this chapter, we have presented the revised version of the Navigator-1 — Navigator-2. This version consists of two parallel sub-nets: SOA and DOA sub-nets, an AND combiner and a decision-making sub-net. In Navigator-2, the SOA sub-net produces a group of static obstacle collision-free commands by directly transferring the local sensory information in the local grid. Meanwhile, a forbidden region is created based on the predicted position and moving direction of a disruptive moving object — if there is such a moving object. The forbidden region often overlaps fully or partially with the local grid. The DOA sub-net thus generates a group of moving obstacle collision-free commands by directly converting the 'virtual sensory information' from the local grid through the network. These two groups of action commands are then combined by using a logical operation AND. Eventually, the decision-making sub-net generates the final collision-free command from the resultant

action commands of the AND combiner. We also described the implementation of each sub-net.

In this chapter, we do not test the Navigator-2 through simulations. This is because the DOA sub-net needs the predictor to provide the prediction for the movement of obstacles. In the next chapter, we will describe a simulation system which implements the whole navigation strategy by integrating three components: planner, navigator and predictor. In that chapter, we are going to examine the behaviour of the Navigator-2.

Chapter 7

Simulation Results

In this chapter, we present simulation results of integrating the three components (planner, navigator and predictor) to navigate a robot in obstacle environments. The navigation simulations are carried out with a simulation system which implements the overall navigation strategy in Turbo C running on a PC386. The purpose of the simulations is to demonstrate that by integrating the three components developed in previous chapters, our navigation strategy is capable of achieving navigation tasks in the presence of unexpected environmental changes. In the following sections, the simulation system itself is introduced first and several navigation cases are then discussed.

7.1 The Simulation System

In order to evaluate the performance of the complete navigation strategy that we proposed, a simulation system has been written using Turbo C on a PC-386. The size of the running program is about 31k bytes. In this system, the three components (planner, navigator and predictor), which we developed in chapter 3, 5 and 6 respectively, collaborate with each other to navigate the robot in a uncertain workspace where unexpected events may occur during navigation. The workspace is simulated as a 40 units by 30 units 2D flat area with fixed known obstacles. The unexpected events include unknown static obstacles and moving obstacles with unknown trajectories. The trajectories of the robot and the moving objects are displayed through animation on the computer's screen so that the performance of the navigation strategy can be observed directly. Note that since we have already examined the capabilities of

each component separately in previous chapters, the purpose of the simulations is to demonstrate how these three components interact with each other. To facilitate the understanding, before introducing the simulation system, we outline again our neural network based navigation strategy in the following subsection.

7.1.1 An overview of the navigation strategy

To tackle the navigation problem in the presence of unpredictable environmental changes, we proposed an autonomous navigation strategy based on neural networks in chapter 1. The purpose of developing the strategy is to give the robot the ability to handle environmental unpredictabilities so that it can be used to perform complex navigation tasks independently in the real world.

The navigation strategy follows a two-level hierarchy, and it is implemented by integrating three network components. At the higher level, the *planner* generates a nominal path according to the map of a workspace which provides the built-in geometric information about the workspace. The nominal path is described as a sequence of collision-free via points (subgoals) from the initial position to the goal position among fixed known obstacles. During navigation, each subgoal provides a temporary direction to drive the robot towards the final goal. Decisions about dealing with unexpected environmental events are delegated to lower-level guidance.

At the lower level, the *navigator* incorporates the *predictor* to achieve on-line guidance by taking into account unexpected environmental changes to refine the coarse path in real-time. When the nominal path is found, the guidance process starts. If a moving object is classified as disruptive, the on-line *predictor* is called to predict the movement of the object one time step ahead in

real-time. Meanwhile, the navigator generates a steering command at each sampling time to avoid collisions with unexpected moving and static obstacles by using the local sensory and prediction information, while driving the robot towards the current subgoal. During this motion, the robot behaves independently from the planner until the planner has to be called to resolve cases that the navigator is unable to handle. In this way, the low-level guidance would be much more autonomous, and the high-level planner would be free from dealing with the environmental uncertainty. Because neural networks have the capabilities of powerful parallel computation, adaptive learning and generalization, they are employed to develop these three components to achieve autonomous navigation. We implemented such a neural network based navigation strategy in a simulation system. Note that the strategy we developed here is a practical/heuristic method that aims to find a navigation solution but does not guarantee to be the shortest path since unpredictable environmental events are involved during navigation. In the next subsection, we describe the implementation of the simulation system.

7.1.2 Implementation of the simulation system

In this subsection, we first introduce the detail of the environmental set-up in the simulation system and then show its control flow chart.

a) Environment set-up

In the system, the robot workspace is simulated as a 2D flat area with three fixed convex obstacles. This area is divided into 40 units by 30 units grid according to the definition of the size of each square in the local grid, and then used as a map while planning a nominal path. The occupied cells in the grid have a state value '1', and the unoccupied cells have a state value '0'.

The unknown static obstacles are represented as polygons which can be put anywhere in the workspace during navigation. To simplify the treatment, we do not concern ourselves with the shape and size of moving obstacles. A moving obstacle is thus represented as a small solid circle with its diameter equal to the width of a cell in the grid. The maximum speed of a moving obstacle is assumed to equal the speed of the robot, which is 1 unit at a time. The robot is simulated as a point moving at a constant speed. The trajectory of the robot is represented by a sequence of occupied cells. In the system, the cases for unknown stationary obstacles and moving objects have been programmed into four environmental configurations. By indicating a corresponding number at the beginning of a navigation, different environmental changes are present during the navigation.

Before a navigation starts, the initial position and the goal position of the robot can be chosen randomly inside the workspace, except at the corners and boundaries. With different goals or start positions, the robot is actually put in different obstacle conditions, and various navigation trajectories are thus obtained.

b) The navigation process

The control flow chart of the system is shown in Figure 7-1. In the system, a navigation process starts by setting the initial position and the goal position, and choosing a configuration for an obstacle condition. Note that if a configuration includes moving obstacles, the moving obstacle indicator is set to 1. When these initial parameters are given, the planner is called first to generate a sequence of subgoals among the fixed known obstacles of the workspace. We implement the planner using Park and Lee's algorithm rather than the enhanced algorithm we developed in chapter 3. This is because the fixed obstacles remain the same at each navigation. Additionally, in the

simulation it would take a long time to have the planner network learn the occupation state of the 40 by 30 grid workspace.

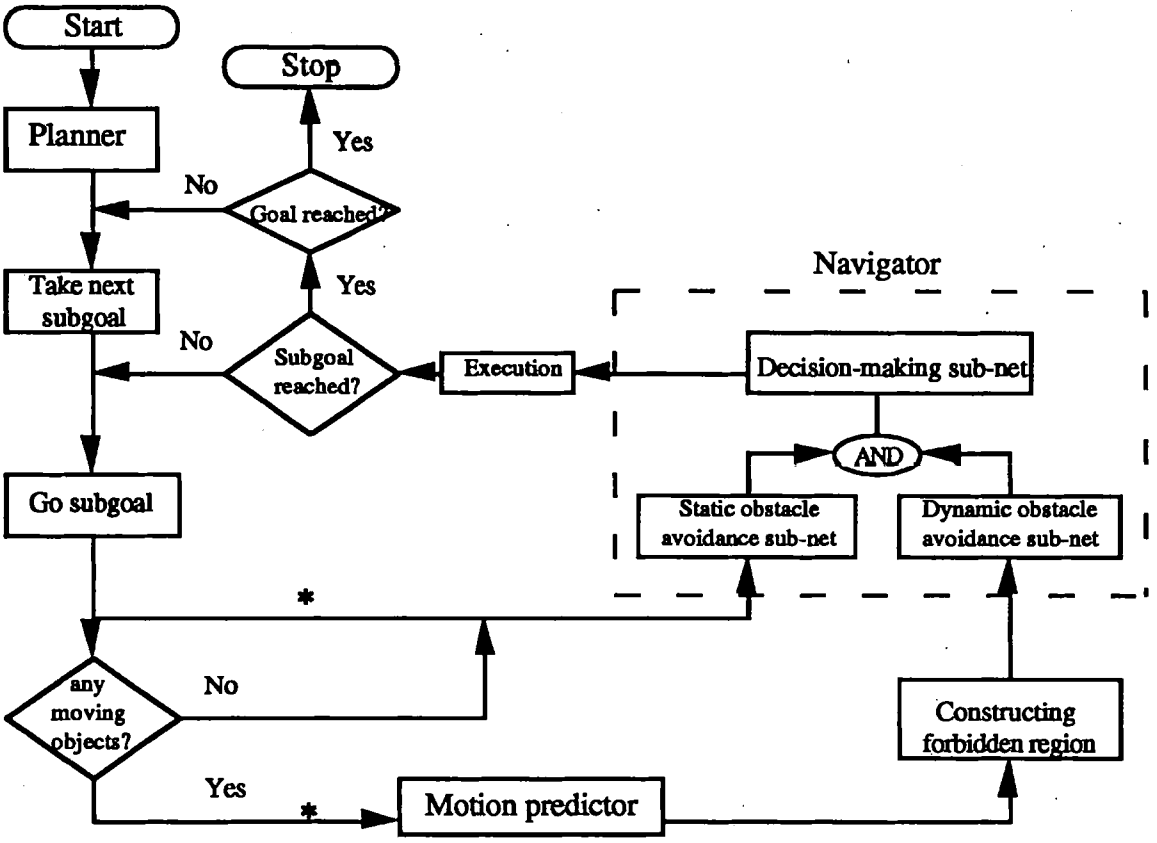


Figure 7-1. Control flow chart of the system (Branches marked with * operate in parallel)

When the subgoals are returned from the planner, the process goes to the 'go subgoal' loop where the first subgoal is taken as the current subgoal of the navigator. The navigator is implemented by directly using the trained sub-nets described in chapter 6. The SOA sub-net of the navigator firstly generates a group of static obstacle collision-free commands by taking sensor readings from the local grid, which was redefined in the section 6.3 of chapter 6. If the state of the moving obstacle indicator is identified as 1, the motion predictor is called to predict the obstacle's movement. The recurrent motion predictor shown in Figure 5-2 of chapter 5 is employed directly here. If the distance between the robot and the predicted position of the moving obstacle is approximately less than or equal to 2.828h, the forbidden regions are created successively at each

sampling time using the method described in subsection 6.4.1 of chapter 6. The sampling time is one execution of the loop. By taking the 'virtual sensory information' from the local grid, the DOA sub-net of the navigator then produces a group of dynamic obstacle collision-free commands. The output commands from both SOA and DOA sub-nets are combined by the AND logical combiner. The resultant combination is further sent to the decision-making sub-net of the navigator which generates the final collision-free steering command. The position of the robot is updated by the executor which moves the robot's position according to the final steering command issued from the navigator. Note that we do not concern ourselves with the problem of low-level servo control in this research, therefore we assume that no execution error exists. The next step is to check if the current subgoal is reached. If it is not, the navigation process goes back to the start of the loop again and repeats the loop until the current subgoal is reached. Otherwise the next subgoal is taken as the current subgoal. The process is repeated until the final subgoal – the goal – is reached. The navigation situation is displayed graphically. The graphic output displays the movements of the objects and the robot through animation on the screen of the computer.

So far, we have described the implementation of the simulation system, in the next section, we present the results of the simulations.

7.2 Simulation Results

The scenario simulated by the system is that a robot navigates along the planned path in the 40 x 30 workspace. During the navigation, some unexpected changes would happen to the workspace (i.e. unknown objects are put at somewhere or an object is moving cross the planned path).

In the following simulation examples, the fixed known obstacles are shown by

solid black areas; the unknown static obstacles are shown by shaded polygons; the current position of a moving object is indicated by a solid circle; and the past positions of the moving object are indicated by a sequence of shaded circles. The subgoals generated by the planner are shown by highlight crosses. The current position of the robot is indicated by a solid black cell, and the past trajectory of the robot is shown by a sequence of shaded cells. In the simulations, unless stated explicitly, we assume that the robot traverses one cell within a sampling interval and the speed of the moving obstacle is never faster than the speed of the robot.

A navigation starts by inputting the x, y coordinates of the robot's initial position, and choosing a configuration of obstacle conditions from the fixed cases 0-3. For the purpose of simplicity, we assign the goal position at (26, 1) for all the cases since users have to choose a goal position with certain constraints. Note that a goal too close to an obstacle cannot be selected, i.e. it should be at least a cell away from any obstacles, otherwise it cannot be reached. Only one subgoal is provided by the path planner since the maximum distance between an initial position and the goal position is at most 28 cells.

To test the effectiveness of the navigation strategy in the presence of a range of unexpected environmental changes, we selected four typical navigation cases:

no environmental changes during navigation;
unknown static obstacles are put on the robot's path;
a moving obstacle with unknown trajectory is crossing the planned path;
both unknown moving obstacle and static obstacles disrupt the motion of the robot at same time.

Additionally, to show the function of the planner in navigations, a special simulation is also conducted in the first navigation example, where the

planner is removed from the system, and the robot is only under the guidance of the navigator. Next, we present the four navigation cases.

Example 1

In this example, the robot is moving from the position (11, 28) to the position (26, 1). The workspace consists of three known fixed obstacles and no unexpected events occur during the navigation. The robot just moves directly to the goal through the subgoal which is located roughly at (18,15). Figure 7-2 shows the intermediate positions of the robot along its trajectory.

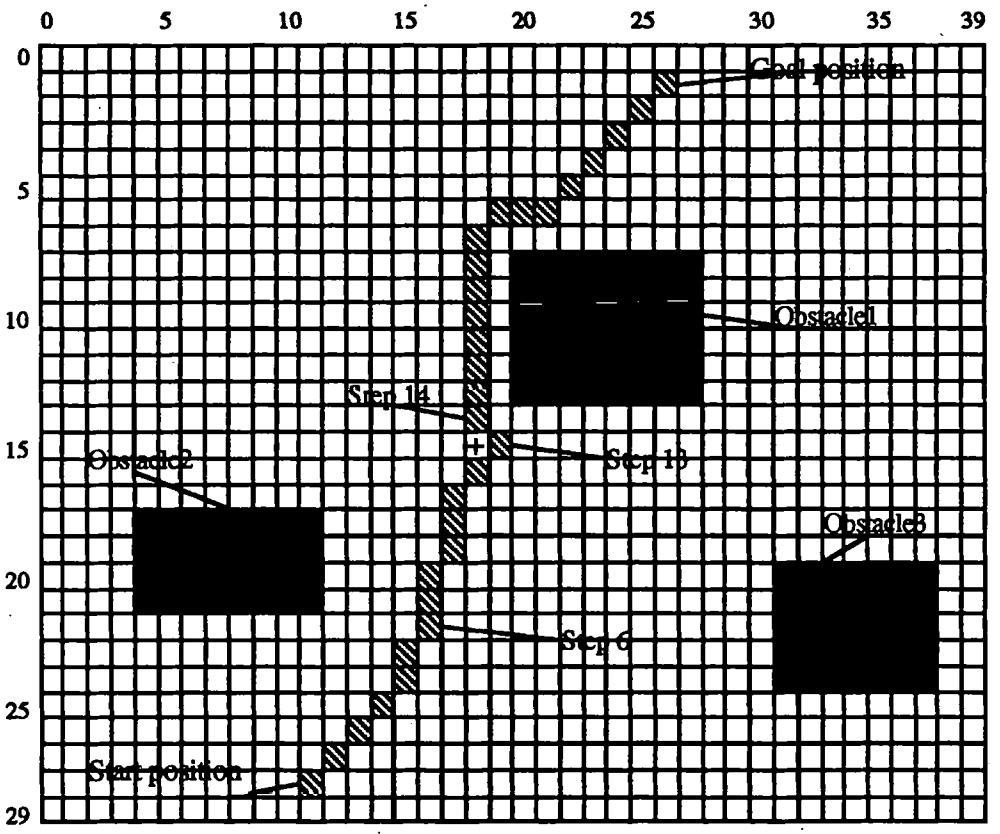


Figure 7-2. Trajectory-1: an ideal navigation case

To evaluate the function of the planner, we then removed the planner from the system and navigate the robot from (11, 28) to (26, 1) again. In this situation, therefore, no subgoal is provided between (11, 28) and (26, 1). The

robot is only under the guidance of the navigator. Figure 7-3 illustrates the new trajectory of the robot. It can be seen that from the figure, the robot starts avoiding the obstacle1 at step 13 since only local information is provided in the local grid, and it therefore needs three steps to find obstacle's edge (i.e step 16). However, with the subgoal (18, 15), the robot need less steps to avoid the obstacle1 as shown in Figure 7-2. This is because the subgoal provides a temporary direction that avoids the collision with obstacle1 and leads the robot towards the final goal when it is not in sight.

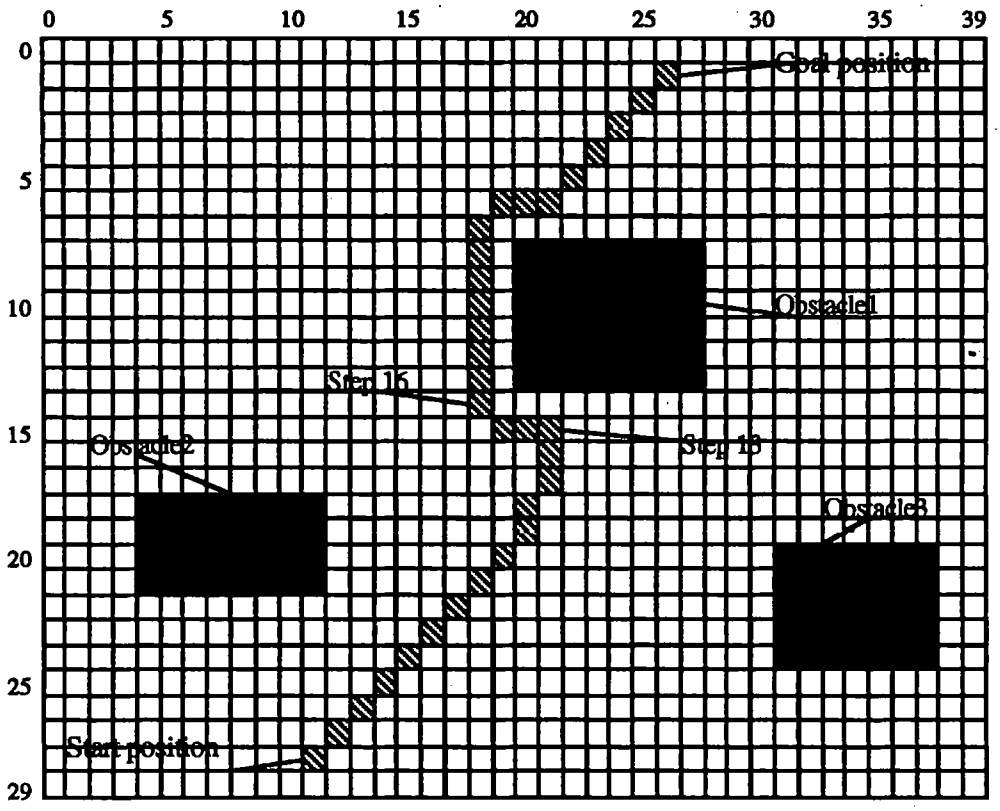


Figure 7-3. Trajectory-2: a navigation without the planner

Example 2

In this example, the robot is also moving from (11, 28) to (26, 1). There are two unknown stationary: obstacle4 and obstacle5, which are put into the workspace during navigation. At the 4th sampling time, the unknown object5 is detected, and the navigator changes its planned path (comparing with the trajectory at

step 6 in Figure 7-2) to avoid the static object. Figure 7-4 thus shows a different trajectory from Figure 7-2, although both cases have the same starting and goal points. Note that since there are no moving obstacles in the environment, the predictor is not used in this case.

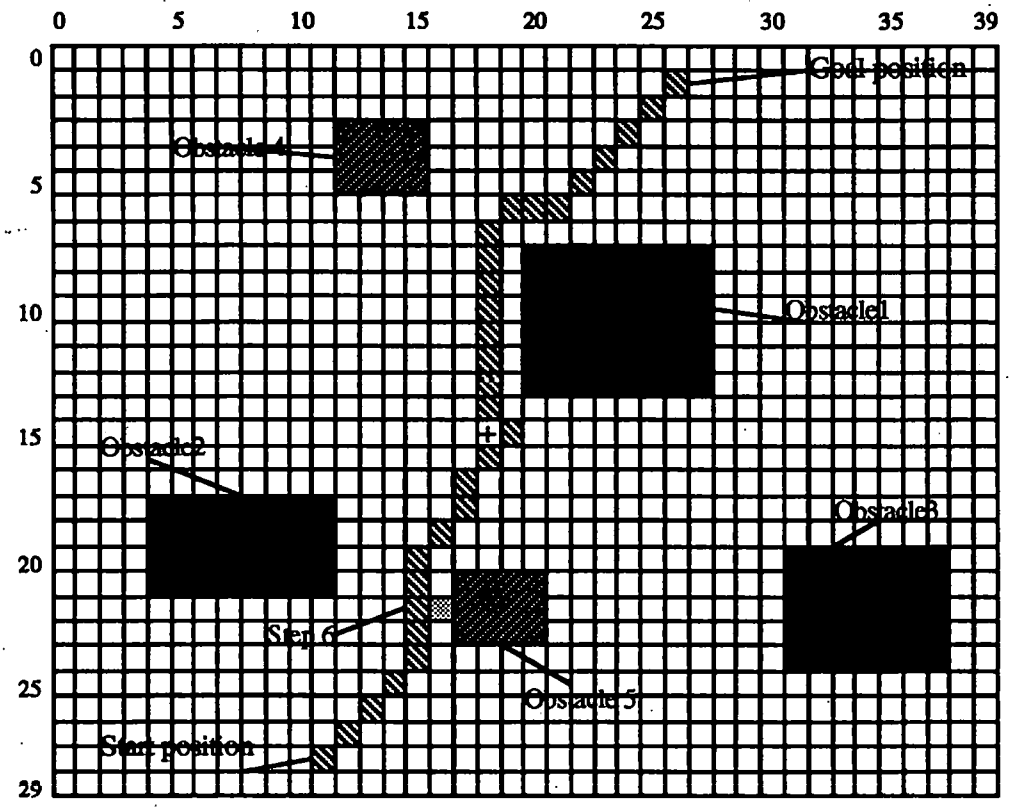


Figure 7-4. Trajectory-3: a navigation with unknown static obstacles

Example 3

In this example, the robot is moving from (11, 28) to (26, 1) again. One object starts at the position (3, 3) and moves cross the planned path. At step 10, the predicted position of the moving object is less than the predefined distance (2.828h) from the robot. The robot starts avoiding the moving object by constructing a fobidden region at each step. At step 13, the moving object is moving away from the robot, and the robot goes back to the planned path. Figure 7-5 shows the trajectory of the robot.

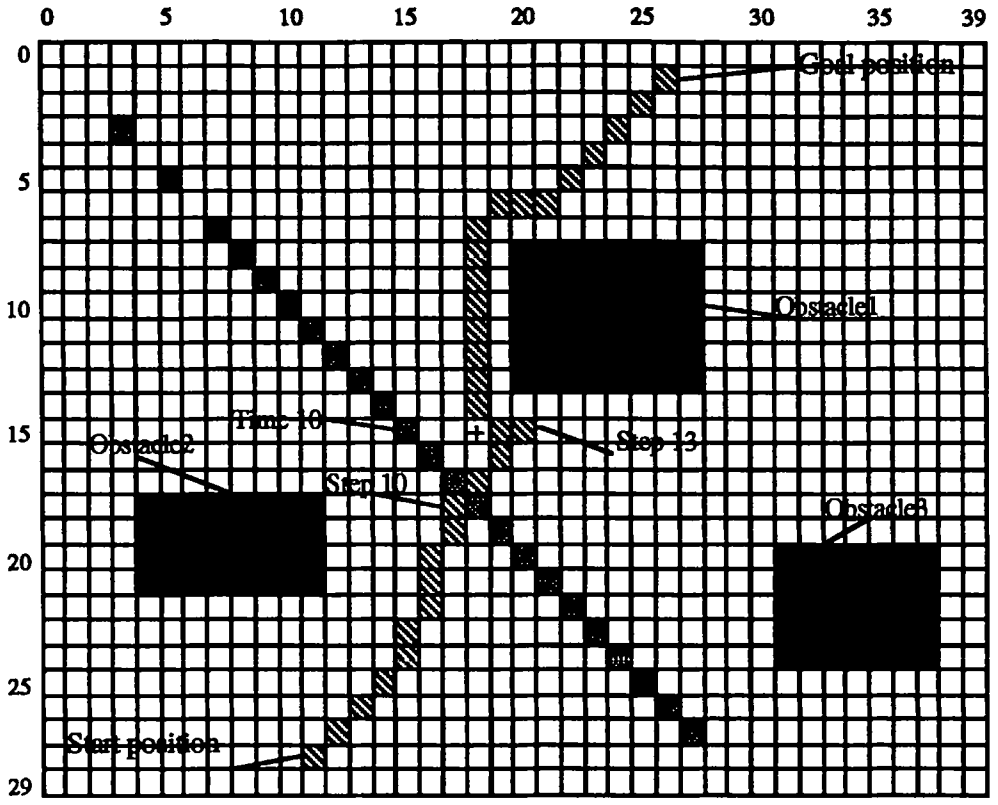


Figure 7-5. Trajectory-4: a navigation with an unknown moving obstacle

Example 4

In this final example, a more complex navigation situation is conducted where the robot has to avoid the unknown static obstacle5 and a moving object at the same time. When the robot is moving from (11, 28) to (26, 1), the unknown stationary obstacles (object4, and object5) are put to the planned workspace, and a moving obstacle is approaching towards the robot. At step 7, the static obstacle and the predicted position of the moving obstacle are both present in the immediate neighbourhood of the robot – in the local grid. The robot moves toward its left (FL) to avoid collisions with the static and the moving obstacle since the static obstacle is on its right, and the predicted moving direction of the moving object is about -45°. At the 9th sampling time, the moving obstacle moves away from the robot, the robot therefore moves directly to the current subgoal. Figure 7-6 shows that the robot avoids collisions

with obstacles by changing its planned path shortly to coping with these local variations.

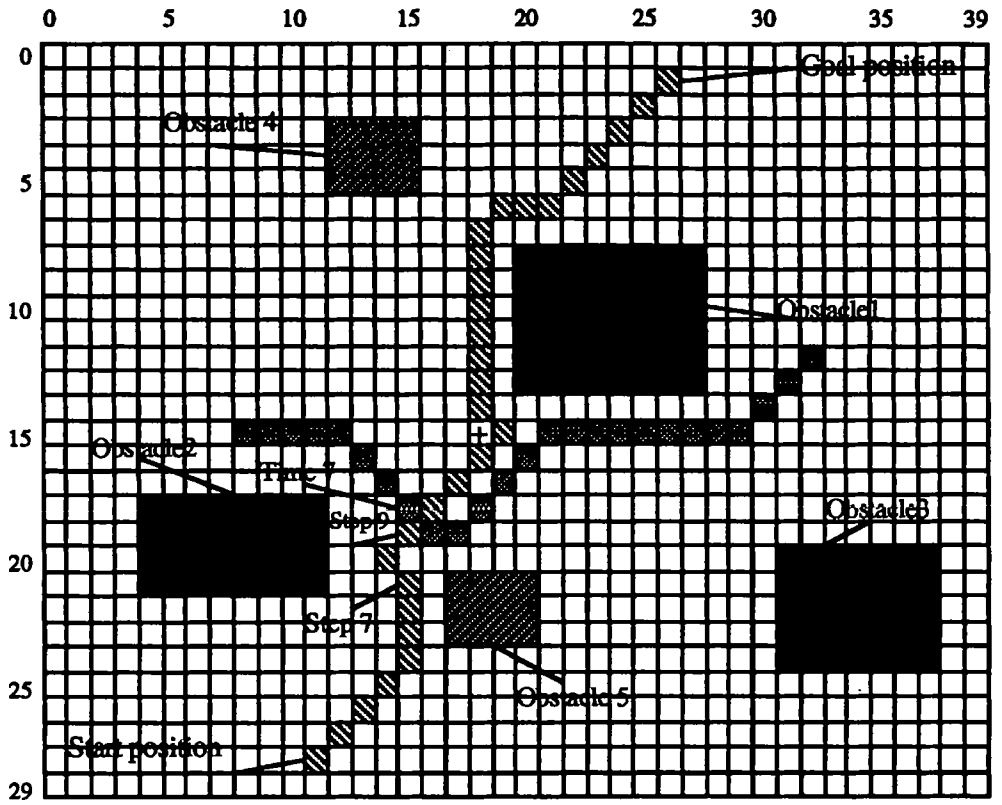


Figure 7-6. Trajectory-5: a complex navigation case

7.3 Summary

In this chapter, we have introduced a simulation system which implements the overall navigation strategy by integrating the three neural network components (planner, navigator and predictor). The purpose of this system is to evaluate the performance of the overall navigation strategy. We showed four typical navigation cases to demonstrate the adaptive capability of the navigation strategy. These four are just a sample of the many simulations that have been tried. Approximately 100 trajectories with different starting positions and different moving obstacles have been simulated with, as yet, no failures. We are therefore confident that this system is reasonably robust under the constraints described in this chapter.

The simulation results show that the two-level hierarchy of the navigation strategy makes effective use of the complementary aspects of a high-level global path planning approach and a low-level local obstacle avoidance approach. By integrating the three network components, the proposed neural network based navigation strategy is capable of handling unexpected events (i.e. unknown static obstacles and moving objects with unknown trajectories) that disrupt the motion of the robot while guiding the robot towards the goal. This can be seen in each of the simulation examples, the robot is always able to reach its goal, and it can handle a range of unexpected events, even in the situation where the movement of the object is almost random (i.e. moving direction).

Chapter 8

Conclusions and Future Work

In this final chapter, we summarise the research work reported in this thesis, and outline its contributions to research in both neural networks and robotics. Finally, we discuss the limitations of this work and present future directions of research that emerge from this study.

8.1 Summary of the Research

The overall goal of this research has been to investigate the feasibility of using neural networks for robot navigation in the presence of unexpected environmental changes. To tackle the navigation problem, we first proposed a neural network based navigation strategy which follows a two-level hierarchy, and operates by integrating three network components (planner, navigator and predictor). Following that, we developed the three neural network components respectively. The path planner was developed first by using a multilayer feedforward network. Then, the first version of the navigator – Navigator-1 – was implemented where the navigation environment was simplified by assuming that only stationary obstacles exist. To enable the navigator to handle moving objects with unknown trajectories, an on-line motion predictor was introduced. The motion predictor was developed using an Elman recurrent net. After implementing the predictor, the improved version of the Navigator-1 – Navigator-2 – was developed which is able to deal with the situations where moving obstacles and (or) static obstacles were involved. Finally, the overall navigation strategy was demonstrated in a simulation system. Simulations were conducted to evaluate the performance

of the navigation strategy, which showed encouraging results. It appears that by integrating the three network components, the overall navigation strategy is capable of achieving navigation tasks adaptively in the presence of unexpected environmental changes.

8.2 Contributions

The contributions of this work are listed below and each of them will be explained in turn:

- **Using a neural network based strategy to solve the robot navigation problem in the presence of unexpected environmental changes;**
- **Improving a neural network based real-time path planning algorithm;**
- **Developing a neural network navigator for on-line local guidance;**
- **Developing a neural network on-line motion predictor.**

8.2.1 Using a neural network based strategy to solve the robot navigation problem in the presence of unexpected environmental changes

In this thesis, an autonomous neural network based navigation strategy has been developed to solve the robot navigation problem in the presence of unexpected environmental changes. To be useful in the real world, mobile robots must be able to deal with unstructured environments (Thorpe, 1991). When a robot performs navigation tasks in an unstructured environment, some unexpected events might occur that can disrupt the motion of the robot. Since real world environments are unpredictable, complete planning for obstacle avoidance cannot be done in advance. Therefore, it is necessary to find an approach to handle unexpected environmental changes during navigation. This research illustrates a novel way to tackle the navigation problem. Neural networks, as a new technique, are used to develop three functional

components (planner, navigator and predictor) which are integrated to achieve autonomous navigation. The navigation strategy we proposed follows a two-level hierarchy. At the higher level, the network *planner* generates a nominal path from the initial position to the goal position among the fixed known obstacles. At the lower level, the *navigator* incorporates the *predictor* to achieve on-line real-time guidance by taking into account unexpected environmental changes to refine the coarse path. During this motion, the robot behaves independently from the planner until the planner has to be called in cases that the navigator is unable to handle. In this way, the low-level guidance would be much more autonomous, and the high-level planner would be free from dealing with the environmental uncertainty. The simulation results described in chapter 7 have shown that the neural network based strategy is capable of achieving navigation tasks adaptively in the presence of unexpected environmental changes.

8.2.2 Improving a neural network based real-time path planning algorithm

During this research, a neural network based real-time path planning algorithm has been developed for the planner to generate a collision-free nominal path. Apart from the functions of conventional planners, the planner used in our navigation strategy should be capable of achieving real-time replanning in case the navigator is unable to handle complex situations due to environmental unpredictability. However, conventional path planning schemes often require some preprocessing and graph searching, and thus are generally computationally intensive and are difficult to implement on-line. This research tackles this problem by employing neural networks which have the capabilities of parallel computation and distributed representation. To adapt the real-time requirements, we developed a neural network path planning algorithm which is an improvement on Park and Lee's (1990) algorithm. In our enhanced method, instead of hand-crafting a number of

networks, the collision penalty function associated with all the obstacles of the workspace is automatically derived by having a backpropagation network to learn the relationship between the coordinate of the workspace and its corresponding occupation state. The energy of a path is then defined using the collision penalty function and the length of the path. By moving the path so that energy is minimised, a collision-free path is obtained at the equilibrium state. Our simulation results showed satisfactory results.

8.2.3 Developing the neural network navigator for on-line local guidance

In this research, two versions of navigator have been developed to guide the robot towards each subgoal provided by the planner, while avoiding unexpected obstacles. During a navigation, unexpected events, which involve unknown static obstacles and moving objects with unknown trajectories, may occur at any time that can disrupt the motion of the robot. Therefore, sensing becomes essential, and sensor-based guidance must be done adaptively and in real-time. In conventional methods, however, sensing and action are treated separately. The sensor-based guidance is often done by pre-establishing an explicit model to represent the relationship between the specific sensor readings and the corresponding steering variables. Problems will arise when unforeseen changes happen to the geometrical parameters of the environment, or to the sensing or mechanical parameter, etc. We tackle this problem using the neural network technique, since neural networks can lend themselves as flexible "function approximators" between sensing and action by adaptation and learning. At first, we developed a navigator – Navigator-1 – in which the navigation environment is simplified by assuming that only stationary obstacles exist. Navigator-1 is built using the proposed neural network based local steering scheme, in which the steering decisions are made by directly converting the local sensory information collected in the local grid through a backpropagation network. The simulation results have shown that

the Navigator-1 can achieve on-line guidance adaptively in the presence of unexpected stationary obstacles.

Following that, we developed an improved version of Navigator-1 – Navigator-2 – which is capable of dealing with situations where moving obstacles are involved. With moving obstacles, the task of the navigator is much more difficult since moving obstacles impose time-varying constraints on the motion of the robot. By constructing a forbidden region at each sampling time to represent these time-varying constraints, the dynamic obstacle avoidance problem can be transferred into a static problem for each time increment. The complicated obstacle avoidance was therefore decomposed into two parallel and independent process: static obstacle avoidance and dynamic obstacle avoidance. Based on the decomposition, we implemented the Navigator-2 using a structured network in which two sub-nets are used to realise dynamic obstacle avoidance and static obstacle avoidance respectively, and the other sub-net is used to make final steering decision by reconciling the results from those two sub-nets. The simulation results presented in chapter 7 have demonstrated that the Navigator-2 is capable of dealing with unexpected events that include not only unknown static obstacles but also the moving obstacles with unknown trajectories.

8.2.4 Developing a neural network on-line motion predictor

During this research, a neural network motion predictor has been developed to predict the movement of the disruptive objects one time step ahead. Predicting the motions of objects in an unconstrained environment is often a difficult task since objects can change their motions in any way and at any time. The information about the movement of a object can only be acquired through processing measurements obtained by sensing devices. In conventional adaptive estimation methods, some linear approximate models

of the unknown process have to be constructed from a sufficiently long sample of sequence data. In the situation of predicting unconstrained motions, however, the underlying behaviour of a moving object varies with time, and the linearity condition may not always be well obeyed. Additionally, the result of prediction at each time will be immediately used by the navigator to avoid the moving object. Therefore, it would be impossible for a navigation system to wait a period until sufficient sampling data has been collected. This research presents a novel approach based on neural networks to tackle this problem, since neural networks have inherent nonlinearity and capabilities of learning and adaptation. The on-line predictor is developed based on an Elman recurrent net, which has the characteristic of dynamic memory. The pseudo-impedance control rule is employed to train the predictor network to get fast convergence. An on-line learning and prediction scheme is developed to adapt predictions with time, which allows the predictor to carry on learning for each trial while making a prediction. The simulations have shown satisfactory results, even when predicting a chaotic time-series.

8.3 Future Work

Though our research has shown some promising results, it has some limitations. In this section, we outline the limitations of this research work and discuss possible future directions for this research.

a) Speed up the learning process in generating the collision penalty function

As we described in chapter 3, the training process for generating the collision penalty function of the workspace needs more than two hundred thousands sweeps of training using the standard backpropagation learning rule. It is necessary to develop a fast on-line learning algorithm for the network planner to generate the new collision penalty function for the changed workspace so

that on-line replanning can possibly be done in practise.

b) Autonomously learning in network navigator for adaptive guidance

Currently, the two versions of navigator (Navigator-1 and Navigator-2) are implemented using a trained network, and learning only being used at the stage of building the navigator. Therefore, the navigator is only able to handle the situations which can be generalised from the training data. In future development, the navigator should be endowed with the capability of learning while performing guidance so that it can be adaptive to any new environments. This may be done by combining teacher controlled and reinforcement learning paradigms, and integrating them into a on-line guidance and learning scheme in which external events can interact with internal states of the system.

c) Generalising the algorithm for constructing forbidden regions

At present, the moving objects are modelled by a point, their shape and size are thus not considered while constructing forbidden regions. To enable the navigation scheme to be used in situations where the moving objects' size and shape matter, it would be necessary to extend the current method of constructing forbidden regions by taking into account the geometrical constraints of moving objects.

d) Multi-step ahead motion prediction

As described in chapter 5, the predictor we developed is used to predict the object's movement one time step ahead. With the one-step prediction, the navigator can merely deal with such moving objects that have constrained velocities (i.e. the maximum speed of a moving object is less or equal to the

speed of the robot). Ideally, the navigator should be capable of handling objects with unconstrained motions. It is therefore necessary to develop a multi-step motion predictor so that the moving obstacles with higher speed than that of the robot can be handled by the navigator during navigation.

e) Developing a new neural network component to deal with kinematic and dynamic constraints

In this thesis, we did not consider the kinematic and dynamic constraints of the system. We assumed that the robot has a servo system that is capable of following any reference trajectories without delays or errors. In practice, however, the servo control problems can hinder the performance of the system in two ways: (1) The robot can not reach the precise goal position; and (2) obstacle avoidance cannot be guarantied. Therefore, developing a neural network based servo control component would be a new task in the future.

f) Developing a subgoal reselection algorithm – mid-level planning

Currently, the unknown stationary obstacles are restricted to be convex polygons. When the concave objects are present during a navigation, the navigator may fail to direct the robot to reach the goal. This is because the navigator is based on a local approach, in which the sensory information collected by the local grid is very simple and local. It may thus miss a viable solution in a sense it cannot see "beyond its use". In addition, in a situation where a very long but thin object lies across the planned path, though the robot can reach its goal eventually, it takes a large number of steps to pass the long object. This is also because the navigator has a very local perspective of the environment. It is thus necessary in the future to develop a subgoal resection algorithm to overcome these problems, rather than always replanning the global path using the planner.

g) Implementing a real robot navigation system using the proposed strategy

Finally, the most rewarding future research would be to implement the navigation strategy presented in this thesis in a real robot navigation system. In the simulation system described in chapter 7, the navigation environment is simplified, and the navigation process is not simulated in real time. To further evaluate the strengths and weaknesses of our approach, it will be necessary to implement a real robot navigation system which can navigate in real environments.

In conclusion, this thesis has shown that neural networks are capable of performing the three major tasks involved in mobile robot navigation, namely path planning, local guidance and prediction. We believe that this may be viewed as one step towards the realization of artificial neural based robot navigation systems.

References

(Papers marked with an asterisk (*) have been produced in the course of this research. All others are referenced in the text of this thesis.)

Arkin, R. C. (1987). "Motor schema-based mobile robot navigation", *Proc. of the IEEE Int. Conf. on Robotics and Automation*, April, 264-271.

Brooks, R. A. (1983). "Solving the find-path problem by good representation of free space", *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-13(3): 190-197.

Brooks, R. A. (1986). "A robust layered control system for a mobile robot", *IEEE J. Robotics and Automation*, RA-2, April, 14-23.

Brooks, R. A. (1991). "Intelligence without representation", *Artificial Intelligence*, 47: 139-159.

Canny, J. and Reif, J. (1987). "New lower bound techniques for robot motion planning problems", *Proceedings of 28th IEEE Symposium on Foundations of Computer Science*, Los Angeles, CA, October, 49-60.

Cheung, J. and Lumelsky, V. J. (1989). "Proximity sensing in robot manipulator motion planning: system and implementation issues", *IEEE Trans. on Robotics and Automation*, 5(6): 740-751.

Desieno, D. (1988). "Adding a conscience to competitive learning", *Proc. IEEE Int. Conf. on Neural Networks*, San Diego, California, July 24-27, 1: 117-124

Elman, J. L. (1990). "Finding structure in time", *Cognitive Science*, 14: 179-211.

Erdmann, M. and Lozano-Perez, T. (1987). "On multiple moving objects", *Algorithmica*, 2(4): 477-521.

Feigenbaum, M. J. (1978). "Quantitative universality for a class of nonlinear transformations", *Journal of Statistical Physics*, 19(1): 25-52.

Feng, D. and Krogh, B. H. (1990). "Satisficing feedback strategies for local navigation of autonomous mobile robots", *IEEE Trans. on Systems, Man, and Cybernetics*, 20(6): 1383-1395.

Fujimura, K. and Samet, H. (1989). "A hierarchical strategy for path planning among moving obstacles", *IEEE Transactions on Robotics and Automation*, 5(1): 61-69.

Fujimura, K. and Samet, H. (1990). "Motion planning in a dynamic domain", *Proc. of IEEE Int. Conf. on Robotics and Automation*, Cincinnati, Ohio, May 13-18, 1: 324-330.

Funahashi, K. C. (1989). "On the approximate realization of continuous mapping by neural networks", *Neural Networks*, 2: 183-192.

Gil de Lamadrid, J. F. and Gini, M. (1990). "Path tracking through uncharted moving obstacles", *IEEE Trans. on Systems, Man, and Cybernetics*, 20(6): 1408-1422.

Gorman, R. P. and Sejnowski, T. J. (1989). "Learned classification of sonar targets using a massively parallel network", *IEEE trans. on Acoustics, Speech and Signal Processing*, 36(7): 1135-1140.

Gouzenes, L. (1984). "Strategies for solving collision-free problems for mobile and manipulator robots", *The International Journal of Robotics Research*, 3(4): 51-65.

Grossberg, S. and Kuperstein, M. (1986). *Neural Dynamics of Adaptive Sensory Motor Control*, Elsevier North-Holland Press.

Hecht-Nielsen, R. (1990). *Neurocomputing*, Addison-Wesley Publication Company.

Hinton, G. E., McClelland, J. L. and Rumelhart, D. E. (1986). "Distributed representations", Rumelhart, and the PDP Research Group ed., *Parallel Distributed Processing*, Vol 1, MIT Press.

Hopfield, J. T. (1982). "Neural networks and physical systems with emergent collective computational abilities", *Proc. Nati. Acad. Sci.*, 79: 2554-2558.

Hopfield, J. J. and Tank, D. W. (1986). "Computing with neural circuits: a model", *Science*, 233: 625-633

Horne, B., Jamshidi, M. and Vadiee, N. (1990). "Neural networks in robotics: a

survey", *Journal of Intelligent and Robotic Systems*, 3: 51-66.

Hornik, M. S. K. and White, H. (1989). "Multilayer feedforward networks are universal approximators", *Neural Networks*, 2: 359-366.

Jahanbin, M. R. and Fallside, F. (1988). "Path planning using a wave simulation technique in the configuration space", *Artificial Intelligence in Engineering: Planning*, J. S. Gero (eds), Elsevier Computational Mechanics Publications.

Jorgenson, C. C. (1987). "Neural network representation of sensor graphs in autonomous robot path planning", *Proc. of IEEE 1st Int. Conf. on Neural Networks*, 4: 507-516.

Josin, G. (1988). "Integrating neural networks with robots", *AI Expert*, August, 50-58.

Kambhampati, S. and Davis, L. S. (1986). "Multiresolution path planning for mobile robots", *IEEE J. Robotics Automation*, RA-2(3): 135-145.

Kant, K. and Zucker, S. W. (1986). "Toward efficient planning: the path-velocity decomposition", *International Journal of Robotics Research*, 5(3): 72-89.

Kant, K. and Zucker, S. W. (1988). "Planning collision-free trajectories in time-varying environments: a two-level hierarchy", *Proc. of IEEE Int. Conf. on Robotics and Automation*, Philadelphia, April, 1644-1649.

Khatib, O. (1985). "Real-time obstacle avoidance for manipulators and mobile robots", *IEEE Int. Conf. on Robotics and Automation*, 500-505.

Kohonen, T. (1984). *Self-organization and Associative Memory*, Springer-Verlag, Berlin.

Kuperstein, M. (1988). "Neural model of adaptive hand-eye coordination for single postures", *Science*, 239: 1308-1311.

Kuperstein, M. (1991). "INFANT neural controller for adaptive sensory-motor coordination", *Neural Networks*, 4: 131-145.

Kyriakopoulos, K. J. and Saridis, G. N. (1990). "On-line motion planning for

mobile robots in non-stationary environments", *Proc. of 5th IEEE Int. Symp. on Intelligent Control 1990*, 2: 694-699.

Kyriakopoulos, K. J. and Saridis, G. N. (1992). "Distance estimation and collision prediction for on-line robotic motion planning", *Automatica*, 28(2):389-394.

Kyriakopoulos, K. J. and Saridis, G. N. (1993). "An integrated collision prediction and avoidance scheme for mobile robots in non-stationary environments", Automatic, 29(2): 309-322.

Lambert, J. and Hecht-Nielsen, R. (1991). "Application of feedforward and recurrent neural networks to chemical plant predictive modeling", *Proc. IEEE 1991 Int. Conf. on Neural Networks*, Seattle, July, 1: 373-378.

Lapedes, A. and Farber, R. (1987). *Nonlinear Signal Processing using Neural Networks: Prediction and System Modeling*, Technical Report No. LA-UR-87-2662, Los Alamos National Laboratory.

Latombe, J. C. (1991). *Robot Motion Planning*, Kluwer Academic Publishers.

Lozano-Perez, T. and Wesley, M. A. (1979). "An algorithm for planning collision-free paths among polyhedral obstacles", *Commun. of ACM*, 22(10): 560-570.

Lozano-Perez, T. (1983). "Spatial planning: a configuration space approach", *IEEE Trans. on Computers*, C-32(2): 26-38.

Lumelsky, V. J. and Stepanov, A. A. (1987). "Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape", *Algorithmica*, 2: 403-430.

Lumelsky, V. J. and Skewis, T. (1988). "A paradigm for incorporating vision in the robot navigation function", *IEEE Int. Conf. on Robotics and Automation*, Philadelphia, PA, April, 734-739.

Massone, L. and Bizzi, E. (1989). "Generation of limb trajectories with a sequential network", *Proc. of Int. Joint Conf. on Neural Networks*, 2: 345-349.

McCulloch, W. S. and Pitts, W. (1943). "A logical calculus of the ideas

immanent in nervous activity", *Bulletin of Math. Bio.*, 5: 115-133.

*Meng, H. and Picton, P. D. (1992). "Obstacle avoidance using a neural network controller and visual feedback", *Preprints of IFAC Symp. on Intelligent Components and Instruments for Control Applications*, May 20-22, Malaga, Spain, 563-568.

*Meng, H. and Picton, P. D. (1992). "A neural network for collision-free path planning", *Artificial Neural Networks 2*, I. Aleksander and I. Taylor (eds), Elsevier North-Holland Publishers, Vol. 1, 591-594.

*Meng, H. and Picton, P. D. (1992). "Planning collision-free paths in time-varying environments", *Proc. of 1st Int. Conf. on Intelligent Systems Engineering*, August 19-21, Edinburgh, UK, 310-315.

*Meng, H. and Picton, P. D. (1993). "A neural network motion predictor", *Proc. of 3th Int. Conf. on Artificial Neural Networks*, May 25-27, Brighton, UK, 177-181.

Moody, J. and Darken, C. (1988). "Learning with localized receptive fields", in Touretzky, D. et al (eds.), *Proc. 1988 Connectionist Models Summer School*, Carnegie Mellon University, San Mateo, CA: Morgan Kaufmann, 133-143.

Nagata, S., Sekiguchi, M. and Asakawa, K. (1990). "Mobile robot control by a structured hierarchical neural network", *IEEE Control Systems Magazine*, April, 69-76.

O'Dunlaing, C. and Yap, C. K. (1985). "A retraction method for planning the motion of a disc", *Journal of Algorithms*, 6(1): 104-111.

Oommen, B. J., Iyenger, S. S., Nageswara, S. V. and Kashyap, R. L. (1986). "Robot navigation in unknown terrains of convex polygonal obstacles using learned visibility graphs", *Proc. of the AAAI-86*, 1101-1106.

Pan, T. J. and Luo, R. C. (1990). "Motion planning for mobile robots in a dynamic environment with moving obstacles", *Proc. of IEEE Int. Conf. on Robotics and Automation*, Cincinnati, Ohio, May 13-18, 1: 578-583.

Pan, T. J. and Luo, R. C. (1991). "On dynamic motion planning problem", *Proc. of IEEE Int. Conf. on Robotics and Automation*, Sacramento, California, April 9-11, 2: 1073-1078.

Park, J. and Lee, S. (1990). "Neural computation for collision-free path planning", *Proc of Int. Joint Conf. on Neural Networks*, San Diego, California, 2: 229-232.

Pomerleau, D. A. (1989). "ALVINN: an autonomous land vehicle in a neural network", *Advances in Neural Information Processing Systems 1*, D. S. Touretzky (ed.) Morgan Kaufmann, 305-313.

Psaltis, D., Sideris, A. and Yamamura, A. (1988). "A multilayered neural controller", *IEEE Control Systems Magazine*, April, 17-21.

Reif, J. and Sharir, M. (1985). "Motion planning in the presence of moving obstacles", *Proc. of the 26th Annual IEEE Symp. on the Foundations of Computer Science*, Portland, OR, October, 144-154.

Rhodes, I. B. (1971). "A tutorial introduction to estimation and filtering", *IEEE Trans. on Automatic Control*, 16(6): 688-706.

Ritter, H. J., Martinetz, T. M. and Schulten, K. J. (1989). "Topology-conserving maps for learning visuo-motor-coordination", *Neural Networks*, 2(3): 159-168.

Ritter, H. J., Martinetz, T. M. and Schulten, K. J. (1990). "Three-dimensional neural net for learning visuomotor coordination of a robot arm", *IEEE Trans. on Neural Networks*, 1: 131-136.

Rumelhart, D. E., McClelland, J. L. and the PDP Research Group. (1986). *Parallel Distributed Processing*. The MIT Press.

Saerens, M. and Soquet, A. (1991). "Neural controller based on back-propagation algorithm", *IEE Proceedings-F*, 138(1), 55-62.

Seshadri, V. (1988). "A neural network architecture for robot path planning", *Proc. of the Second Int. Symp. on Robotics and Manufacturing: Research, Foundation, and Applications*, ASME Press, 249-256.

Sharir, M. (1989). "Algorithmic motion planning in robotics", *IEEE Computer*, 22(3): 9-19.

Shih, C. L., Lee, T. T. and Gruver, W. A. (1990). "Motion planning with time-varying polyhedral obstacles based on graph search and mathematical

programming", *Proc. of IEEE Int. Conf. on Robotics and Automation,* Cincinnati, Ohio, May 13-18, 1: 331-337.

Slack, M. G. and Miller, D. P. (1987). "Path planning through time and space in dynamic domains", *Proc. of Int. Joint Conf. on Artificial Intelligence,* 1067-1070.

Sparks, D. L. and Nelson, J. S. (1987). "Sensory and motor maps in the mammalian superior colliculus", *Trends in Neurosciences,* 10: 312-317.

Szu, H. (1986). "Fast simulated annealing", *American Institute of Physics,* 420-425.

Thorpe, C. E. (1991). "Mobile robots", *International Journal of Pattern Recognition and Artificial Intelligence,* 5(3): 383-397.

Tolat, V. V. and Widrow, B. (1988). "An adaptive 'broom balancer' with visual inputs", *Proc. of IEEE Int. Conf. on Neural Networks,* San Diego, California, July, 2: 641-647.

Tournassoud, P. (1986). "A strategy for obstacle avoidance and its application to multi-robot systems", *Proc. of IEEE Int. Conf. on Robotics and Automation,* San Francisco, CA, April, 1224-1229.

Tsutsumi, K., Katayama, K. and Matsumoto, H. (1988). "Neural computation for controlling the configuration of 2-Dimensional Truss Structure", *Proc. of IEEE Int. Conf. on Neural Networks,* San Diego, California, July, 2: 575-586.

Tsutsumi, K. (1989). "A multi-layered neural network composed of backprop. and Hopfield nets and internal space representation", *Proc. of Int. Joint Conf. on Neural Networks,* Washington D.C., June, 1: 365-371.

Waibel, A. (1989). "Modular construction of time-delay neural networks for speech recognition", *Neural Computation,* 1: 39-46.

Walter, J., Ritter, H. and Schulten, K. (1990). "Non-linear prediction with self-organizing maps", *Proc. of the Int. Joint Conf. on Neural Networks,* Calif., June, 1: 589-594.

Wasserman, P. D. (1989). *Neural Computing: Theory and Practice.* Van Nostrand Reinhold.

Werbos, P. J. (1989). "Backpropagation and neurocontrol: a review and prospectus", *Proc. of IEEE Int. Joint Conf. on Neural Networks,* Washington D.C., June 18-22, 1: 209-216.

Widrow, B. and Hoff, M. E. (1960). "Adaptive switching circuits", *WESCON Conv. Record,* 4: 96-104.

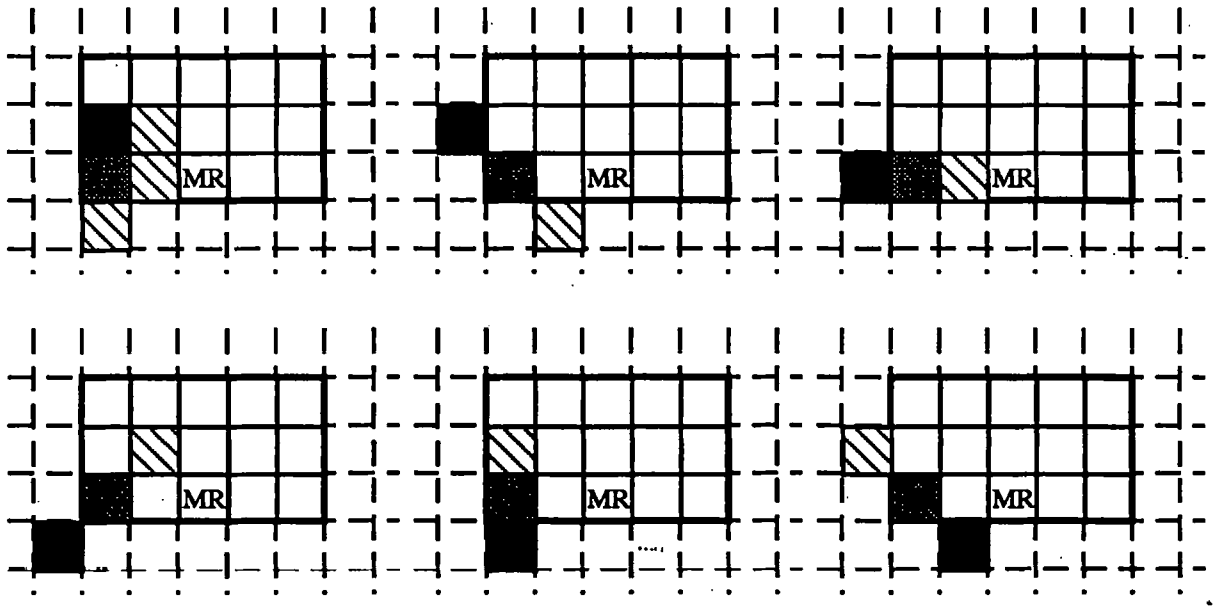
Williams, R. J. and Zipser, D. (1990). *Gradient-Based Learning Algorithms for Recurrent Connectionist Networks,* Northeastern university, Boston, Technical Report NU-CCS-90-9.

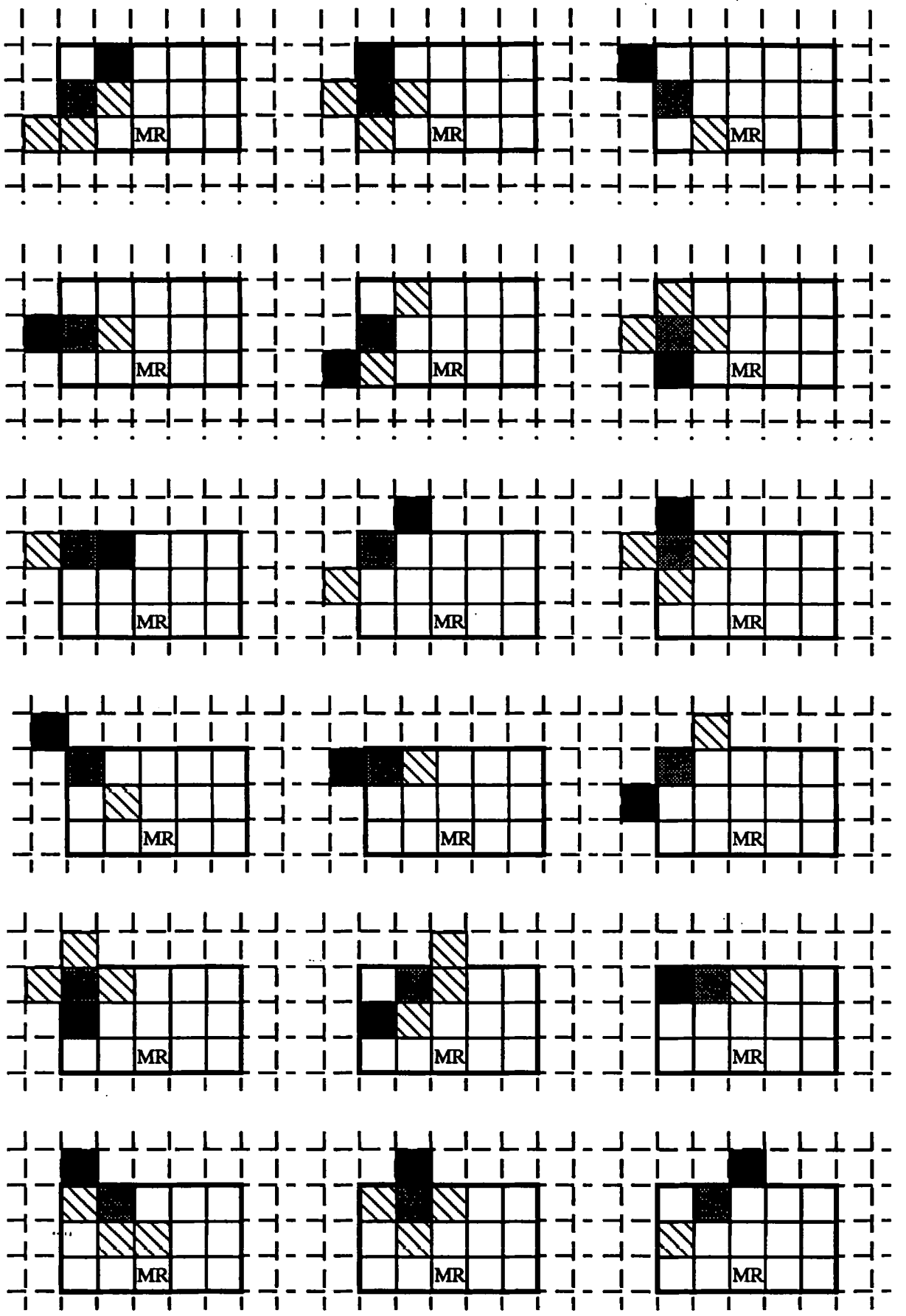
Wilson, G. and Pawley, G. (1988). "On the stability of the travelling salesman problem of Hopfield and Tank", *Biological Cybernetics,* 58: 63-70.

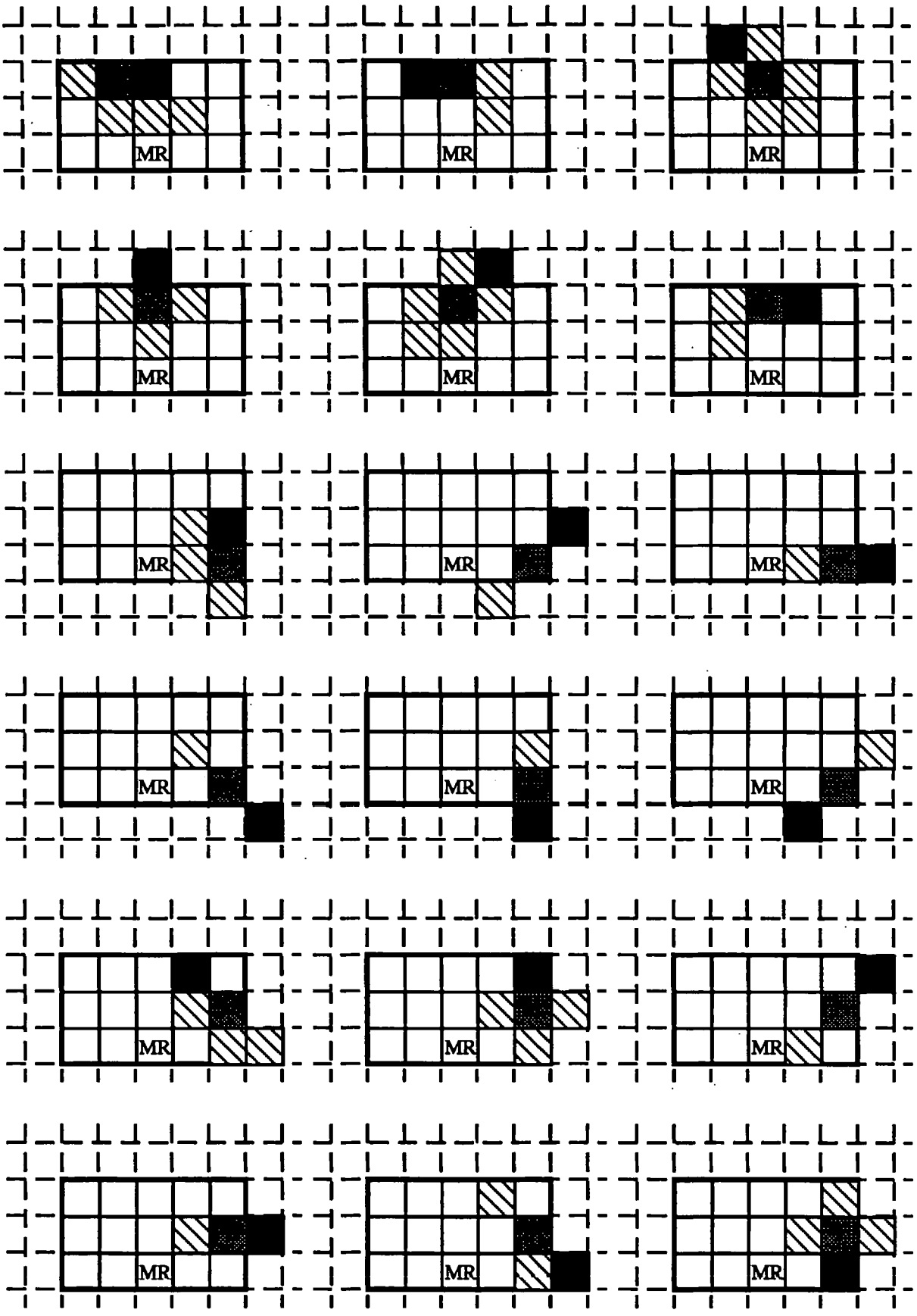
Appendix A

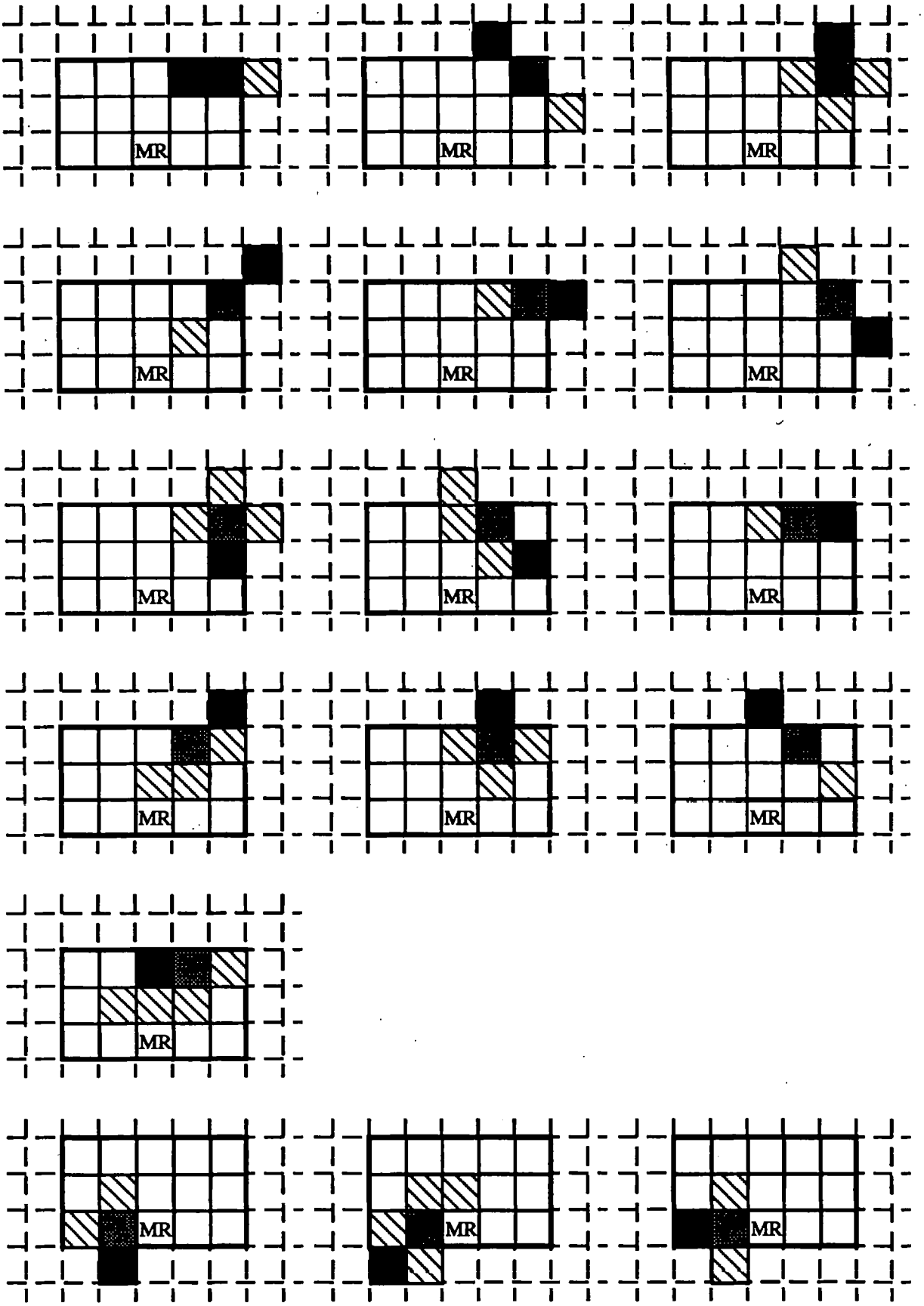
Constructing Forbidden Regions

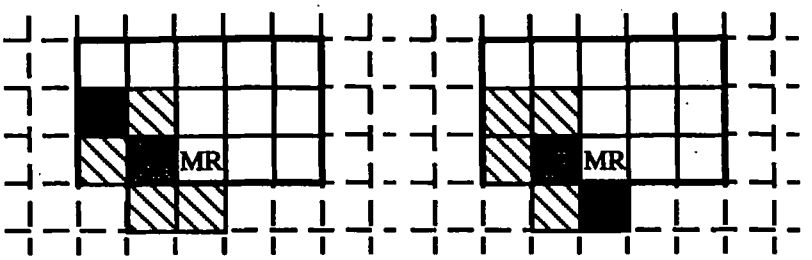
The concept of a forbidden region is introduced to represent the time-varying constraints that are imposed on the motion of the robot by a moving obstacle. The forbidden region at any time is determined in terms of the predicted position, the predicted moving direction and the current position of the moving object. Briefly, the forbidden region is constructed by connecting the current position to the predicted position, and then extending the area along the predicted moving direction to a predefined distance. Specifically, a forbidden region consists of the predicted occupied cell, the current occupied cell and the enlarged part, which is created for the safety purpose. According to the predicted position and the current position of a moving object in the local grid, there are in total 78 cases which are listed below. The current position is illustrated by a solid black cell; the predicted position is shown by a heavily shaded cell; the enlarged part of a forbidden region is illustrated by lightly shaded cells, and MR indicates the current position of the robot. Note that the current position of a moving object cannot be at any first-level squares in the local grid since they are the obstacle-free positions as assumed in chapter 4.

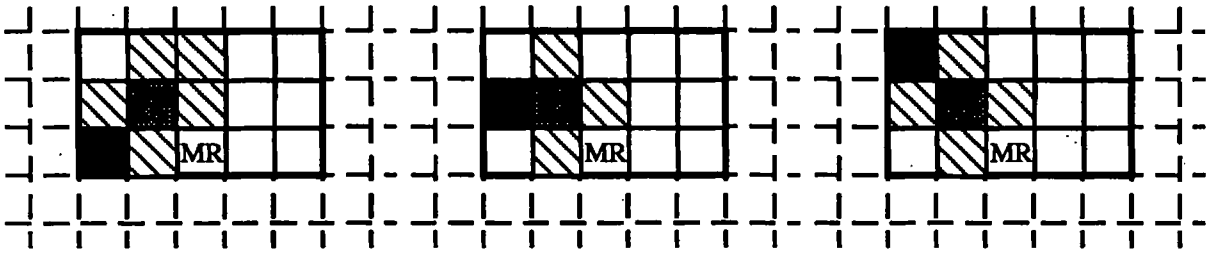


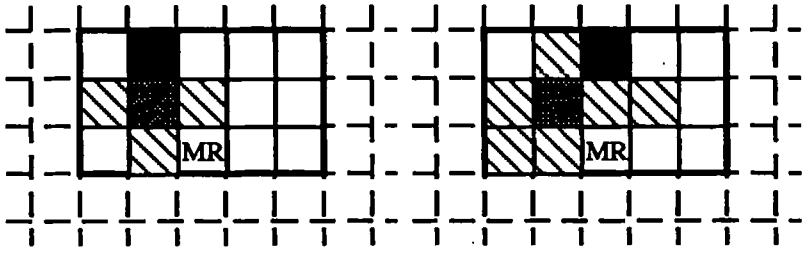

MR
MR
MR
MR
MR
MR
MR
MR
MR
MR
MR
MR
MR
MR
MR
MR
MR
MR

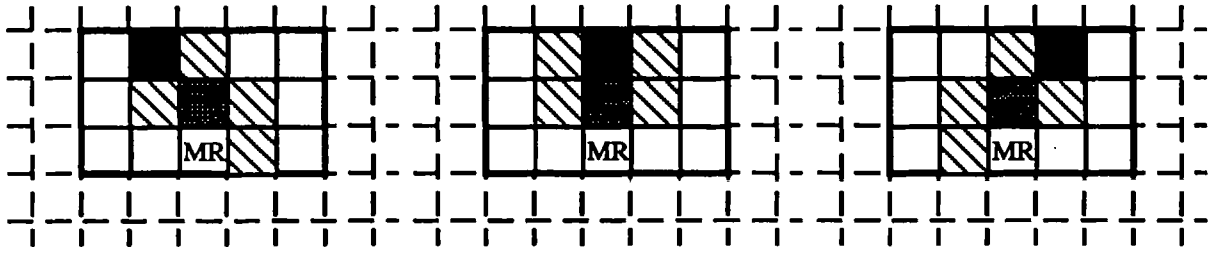

MR
MR
MR
MR
MR
MR
MR
MR
MR
MR
MR
MR
MR
MR
MR
MR
MR
MR
MR
MR
MR

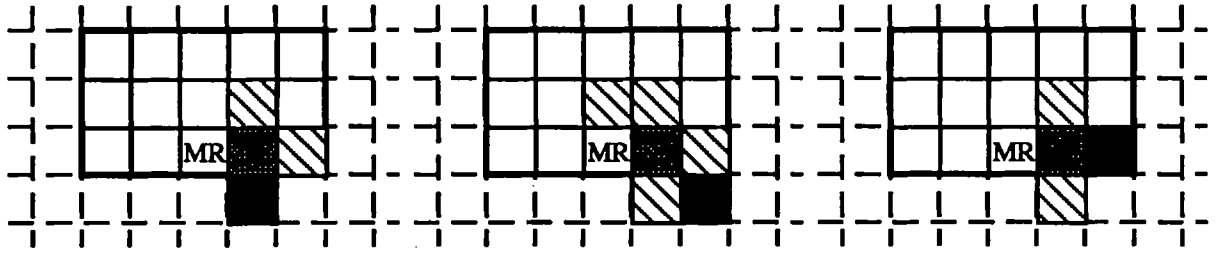

MR
MR
MR
MR
MR
MR
MR
MR
MR
MR
MR
MR
MR
MR
MR

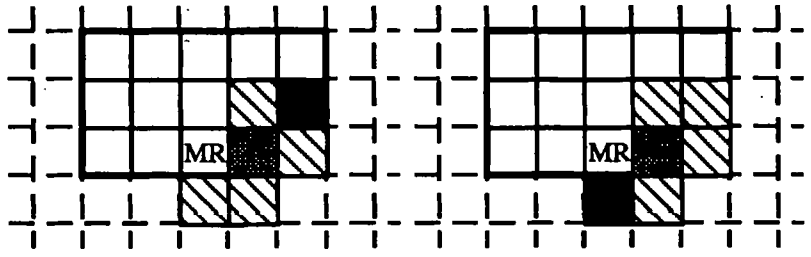

MR
MR

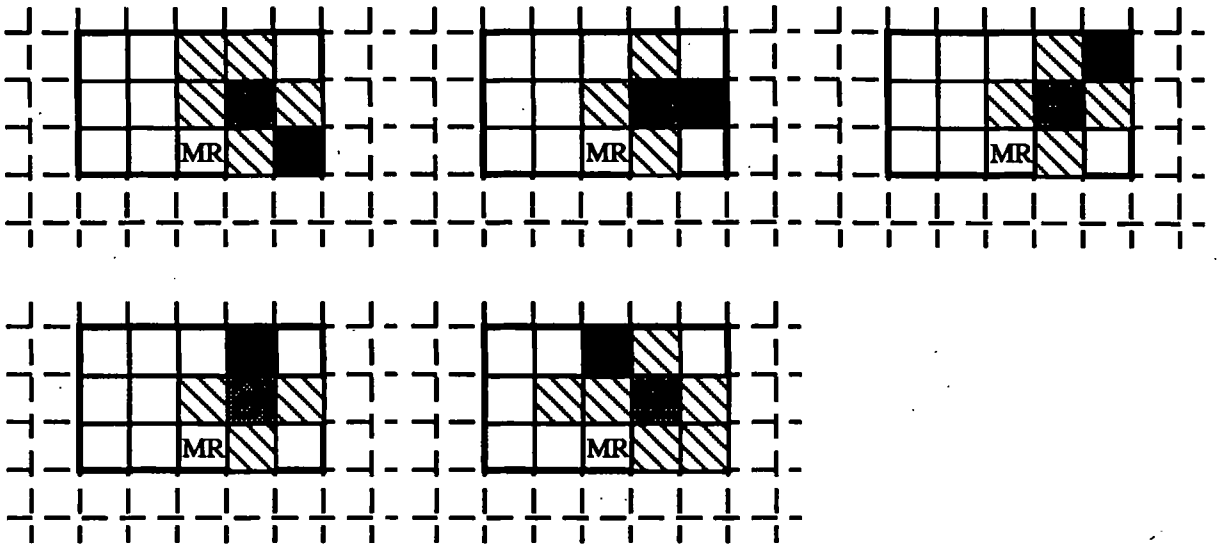

MR
MR
MR


MR
MR


MR
MR
MR


MR
MR
MR


MR
MR


MR
MR
MR
MR
MR